UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Kragelj

FPGA IMPLEMENTACIJA OSNOVNEGA CEVOVODA KODIRNIKA VIDEO KODEKA DAALA

MAGISTRSKO DELO

Mentor: prof. dr. Branko Šter

Ljubljana, 2016

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Večna pot 113
1000 Ljubljana, Slovenija
telefon: 01 47 98 100
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Številka: 160-MAG-RI/2016
Datum: 06. 04. 2016

Peter KRAGELJ, univ. dipl. inž. rač. in inf.

L j u b l j a n a

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **FPGA implementacije osnovnega cevovoda kodirnika video kodeka Daala**

FPGA implementation of basic pipeline encoder of video codec Daala

Tematika naloge:

Razvoj video kompresije je potekal iterativno brez večjih bistvenih sprememb. Predvsem osnovna arhitektura cevovoda za kodiranje slike je ostajala enaka, dodajale so se le naprednejše funkcije. Bistveni gradnik cevovoda je bila diskretna kosinusna transformacija, njeno uporabo pa zasledimo v praktično vseh sodobnih kompresijskih procesih.
Odprtokodni kodek Daala predstavlja pomemben korak v razvoju video kompresije, saj preizkuša uporabo drugačne, diskretne kosinusne transformacije s prekrivajočimi bloki. Pri razvoju kompresijskega algoritma je pomemben vidik tudi zmožnost izvedbe algoritma v strojni obliki v FPGA ali ASIC čipih.
V svoji magistrski disertaciji pripravite strojno implementacijo osnovnega cevovoda kompresije video kodeka Daala v jeziku SystemVerilog. Svojo implementacijo preizkusite na FPGA razvojni plošči. Preučite prednosti in slabosti te implementacije. Podajte rezultate implementacije, zahtevano količino resursov ter njeno zmogljivost. Prav tako podajte predloge za morebitne izboljšave s stališča strojne implementacije.

M e n t o r :

prof. dr. Branko Šter

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

D e k a n :

prof. dr. Nikolaj Zimic

Zahvala

Na prvem mestu bi se rad zahvalil svojemu mentorju, prof. dr. Branku Šteru, ki me je svetoval pri pripravi tega magistrskega dela. Nato bi se rad zahvalil svoji ženi Veroniki in hčerkicama Emi in Zoji, ki so mi vedno stali ob strani. Nazadnje pa bi rad izrekel zahvalo tudi vsem prijateljem in sodelavcem, ki so kakorkoli pripomogli pri izdelavi tega diplomskega dela.

Za sestro Karmen ...

Kazalo

Seznam uporabljenih kratic in simbolov

- CABAC - Algoritem za binarno aritmetično kodiranje
- CIF - standard za resolucijo video vsebin 352x288
- DC koeficient - glavni koeficent bloka
- DCT - diskretna kosinusna preslikava
- FPGA - programabilno logično vezje
- HEVC - ime naprednega kompresijskega standarda
- IDCT - inverzna diskretna kosinusna preslikava
- IEC - mednarodno telo za standardizacijo na področju elektrotehnike
- IP - predpripravljena logična funkcija
- ISDN - komunikacijski standard za prenos zvoka, videa in podatkov
- ISO - mednarodno standardizacijsko telo
- ITU - mednarodno telo za standardizacijo telekomunikacij
- LUT - iskalna tabela
- MPEG - mednarodno telo za standardizacijo
- NTSC - video standard prikaza na analognih TV sprejemnikih v ZDA
- PAL - video standard prikaza na analognih TV sprejemnikih
- PCIE - serisko vodilo
- QCIF - standard za resolucijo video vsebin 176x144
- RAM - pomnilnik z naključnim dostopom
- RDO - optimizacija glede na stopnjo popačenja izhodne slik
- ROM - bralni pomnilnik
- SMPTE - mednarodna organizacija inženirjev s področja televizije in filma

Povzetek

Prikazovanje video vsebin nas spremlja skorajda na vsakem koraku - od TV oddaj na televizijskih sprejemnikih, video vsebin na internetu do celovečernih filmov v kinodvoranah. Zato ni presenetljivo, da se je skozi zgodovino računalništva razvila množica različnih kodirnikov videa, ki so doprinesli k novostim v procesiranju video vsebin.

Video kodirnik Daala je eden izmed novejših video kodirnikov, ki so še v stopnji razvoja. Prinaša nov razvoj video kompresije z izbiro nestandardnih algoritmov za procesiranje. Za prehod v frekvenčno domeno uporablja Haarovo preslikavo in diskretno kosinusno transformacijo (DCT) s prekrivajočimi bloki, za kodiranje izhodnih podatkov pa podvrsto aritmetičnega kodiranja.

V tem magistrskem delu je bila zasnovana in izvedena implementacija osnovnega cevovoda video kodirnika Daala. Namen dela je bil ugotavljanje zmožnosti implementacije in iskanje morebitnih pomanjkljivosti kodirnika Daala. Implementacija je bila osnovana za uporabo na čipu FPGA, kjer je bila tudi praktično preizkušena. Sama implementacija je bila uspešno dokončana. Ugotovili smo, da uporablja predvideno število virov na čipu FPGA, vendar pri tem ne dosega želenih hitrosti, predvsem zaradi neoptimalne izvedbe.

Izkazalo se je, da je osnovni cevovod kodirnika Daala primeren za uporabo na čipih FPGA, saj ne vsebuje gradnikov, ki jih ni mogoče implementirati. Pri tem so se pokazale tudi določne omejitve - algoritmi potrebujejo za svoje delovanje podatke iz celotnega bloka, zato je potebno shraniti podatke tega bloka med posameznimi koraki cevovoda.

Kodirnik Daala je še vedno v fazi razvoja in še ni pripravljen za splošno uporabo, saj določene funkcionalnosti ne delujejo. Razbitje superbloka na bloke brez uporabe optimizacije RDO (Rate-distortion optimization) ne deluje, saj se razvoj kodirnika trenutno izvaja samo z uporabo optimizacije RDO. Ravno tako razvoj aritmetičnega kodiranja ni zaključen, zato je na voljo preveliko število metod kompresije, kar po nepotrebnem zaplete implementacijo.

V tem delu predstavljena implementacija osnovnega kodirnika video kodeka Daala dokazuje možnost njegove praktične implementacije in je prvi korak k polni in bolj optimizirani različici implementacije, ki bi bila primerna tudi za splošno uporabo.

Ključne besede:

osnovni cevovod, kodirnik, video, Daala, implementacija, Haarova preslikava, aritmetično kodiranje

Abstract

Almost everyone in the world uses video compression many times each day - whether when watching show on TV, clips on the Internet or movies at the cinema. So it is not a surprise that through the history of computer science a number of different video codecs have been developed, each bringing some novel approaches to video processing.

Daala video codec is one of the newest video codecs still in development. It is important as it uses nonstandard algorithms for video processing. It uses the Haar transform and the DCT lapped transform for crossing into frequency domain and type of arithmetic coding for coding of output data.

In this work a hardware implementation of basic pipeline of video codec Daala has been developed. The aim of this work was to study possibilities of such implementation and to find possible shortcomings.

Implementation has been prepared for FPGA chip where it was tested. The implementation has been finalized and has been found to use appropriate number of FPGA resources, but does not meet intended speed because of sub-optimal implementation.

It has been showen that basic pipeline of video codec Daala is appropriate for usage in FPGA chips as it does not use building blocks that are impossible to implement in hardware. It has been also showen that there are some limitations. Algorithms need full data of the whole superblock to function, so all data need to be stored for each step of the pipeline.

Video codec Daala is still in development and such not appropriate for the general usage as some functions of the codec are not working. Splitting of superblock into blocks is not working if RDO optimization is disabled as current development is using this optimization. Development of arithmetic coding is also not finished because of this large number of compression methods that are available and which complicate implementation.

This implementation of basic pipeline of video codec Daala confirms possibility of its practical implementation and is a first step to full and more optimized implementation that would be appropriate for general usage.

Keywords:

basic pipeline, encoder, video, Daala, implementation, Haar transform, arithmetic coding

Poglavje 1

Uvod

1.1 Pregled razvoja video procesiranja

Večina ljudi sodobnega sveta vsakodnevno uporablja video kompresijo. Določeno vrsto kompresije namreč uporabljajo vse video vsebine, pa naj si bo to najljubša kuharska oddaja na televiziji, zabavni izseki filmov na internetu ali celovečerni film na DVD predvajalniku.

Leta 1929 je Ray Davis Kell opisal prvi način video kompresije v smislu rešitve prenosa prevelikih količin podatkov in zanj prejel patent. Zapisal je: »V preteklosti je bil običajen način prenosa videa v obliki polnih zaporednih slik. . . /. . . V skladu s tem izumom pa se tej težavi izognemo tako, da prenašamo samo razlike med posameznimi slikami v zaporedju.« Čeprav je od tega zapisa minilo precej časa, preden se je ta tehnika dejansko uporabila v praksi, je Kellov patent še vedno eden izmed osnovnih gradnikov modernih video kompresijskih standardov.

1.1.1 Začetki

Analogno predvajanje

Prvi sistemi za predvajanje videa so se razvili iz osciloskopov. Pri tem je šlo v bistvu za razširitev osciloskopov v drugo dimenzijo, tako da so le-ti lahko prikazovali več črt na zaslonu. Ti sistemi so bili analogni, uporabljali so zaslone s katodno cevjo. Kmalu je sledilo spoznanje, da so takšni zasloni primerni za predstavitev signala, ki se prenaša preko radijskih frekvenc. Tako je nastala prva televizija. Barvna slika je sledila kasneje, ker pa je bila skladnost s črnobelo televizijo nujna, je bil signal zasnovana tako, da je podpiral tako barvno

kot črno-belo modulacijo [4].

Prepletanje

Svetlobno sevanje luminiscenčnega materiala v zgodnjih analognih zaslonih ni bilo dovolj dolgotrajno, da bi omogočalo prikaz čiste slike pri majhnih frekvencah. Za rešitev te težave so strokovnjaki razvili eno najpomembnejših oblik zgodnje oblike kompresije - metodo prepletanja, kjer se izmenjujoče prenašajo lihe in sode vrstice.

Tako izmenjujoče prenašanje vrstic je koristno na dva načina – s tem lahko dejansko prepolovimo zahtevano količino prenosa ali pa z njim podvojimo vertikalno resolucijo. S prenašanjem vsake druge vrstice je zaslon lahko delno osvežil sliko znotraj zahtevanega časa.

Ta in podobne metode se nanašajo na tendence človeškega očesa in možganov, da sestavljajo sliko v koherentno celoto. Zaradi preproste uporabe in velikega učinka je prepletanje postalo standard za vse oblike videa in se uporablja še danes, čeprav večinoma le še zaradi zahtev z združljivostjo [4].

Snemanje

Video zapis na magnetne trakove je prva posnela televizijska hiša BBC leta 1955. Video kasete za končne uporabnike so nastale kasneje, v sedemdesetih letih 20. stoletja, množično pa so se začele uporabljati v osemdesetih letih. Takrat sta nastala tudi prva standarda - Beta in VHS, od katerih se je v splošni uporabi najbolj uveljavil standard VHS.

Istočasno z oblikovanjem standardov je začela nastajati tudi zakonska podlaga za snemanje video vsebin, ki jo je sprožila tožba televizijskih ponudnikov, ki so s tem želeli preprečiti prodajo video snemalnikov. Vrhovno sodišče ZDA je razsodilo, da je snemanje za lastno uporabo dovoljeno in legalno [11].

1.1.2 Digitalno procesiranje

Digitalno video procesiranje se je razvijalo skupaj z razvojem računalniške industrije. Vsak izmed korakov predvajanja, prepletanja in snemanja je dobil tudi svojo digitalno različico. Obstaja večje število standardov za video kompresijo, zato bomo v nadaljevanju opisali le najpommebnejše.

H.120

Leta 1984 je International Telecommunication Union (ITU) standardiziral predlog H.120. To je bil prvi standard za digitalno video kompresijo.

H.120 je uporabljal diferencialno pulzno kodno modulacijo (DPCM), skalarno kvantizacijo in metode kodiranja variabilnih dolžin za kodiranje in prenos videa NTSC in PAL preko zanj določenih komunikacijskih linij. Zahteval je podatkovno širino 1544 kbit/s za vedeo NTSC in 2048 kbit/s za video PAL.

Cilj uporabe tega in vseh sledečih standardov izpod okrilja ITU je bil video telefonija in video konferenca.

Leta 1988 je bila izdana druga enačica standarda, ki je podpirala kompenzacijo gibanja in predikcijo ozadja. Standard je bil pomemben korak na poti k modernim kompresijskim metodam, saj se je izkazalo, da obdelava vsake posamezne točke ni učinkovita.

Nasledniki tega standarda so procesirali slike v blokih. Standard H.120 je v okviru moderne tehnologije zastarel, zato se praktično ne uporablja več [13].

H.261

Praktična uporaba video kompresije se je začela z ITU standardom H.261 iz leta 1990. Cilj tega standarda je bil prenos videa preko liniji ISDN. Omogočal je prenos količine podatkov, ki so bile večkratnik 64kbit/s in resolucijo CIF (352x288) in QCIF (176x144). Standard je bil pionirski dosežek, saj uporablja hibridno metodo za kodiranje videa, ki je osnova za mnogo modernih video kompresijskih standardov.

Hibridno kodiranje videa vsebuje dve metodi:

- Gibanje s slike na sliko je ocenjeno in kompenzirano s podatki s preteklih slik.
- Razlika, ki je pridobljena s to predikcijo, je kodirana s prehodom iz prostorske v 2D frekvenčno domeno. Podatki iz frekvenčne domene so nato kvantificirani (kar predstavlja korak, kjer prihaja do izgube natančnosti) in nato brezizgubno zakodirani s kompresijsko metodo – npr. s Huffmanovim kodiranjem.

H.261 uporablja metodo vzorčenja podatkov 4:2:0, ki vsebuje dvakrat več podatkov o svetlosti kot o barvi, kar se sklada s sposobnostjo človeškega očesa, ki je bolj občutljivo na svetlobne kot na barvne odtenke.

Standard uporablja 16x16 pik velike makrobloke za ocenjevanje gibanja in 8x8 pik velike bloke za diskretno kosinusno transformacijo (DCT), kvantizacijo,

razvrstitev cikcak in Huffmanovo entropijsko kodiranje. Standard deluje na podatkovni širini 64-22048kbit/s.

Standard H.261 je še vedno v uporabi, vendar ga je v veliki večini primerov zamenjal novejši standard H.263. Organizacija in proces za razvoj standarda H.261 je postal osnovna struktura za razvoj vseh nadaljnjih standardov [13].

Motion JPEG

JPEG je ena izmed najbolj razširjenih metod za kompresijo slik. Nastal leta 1992. Standard je široko uporabljen v digitalnih fotoaparatih, mobilnih telefonih in internetu.

Motion JPEG kodira video kot zaporedje neodvisnih JPEG slik. Ker pri tem ne pride do kompenzacije gibanja, ostaja časovna redundanca, ki je osnovna značilnost videa, neizkoriščena. Posledično je razmerje kompresije manjše, kot ga je mogoče doseči s predikcijo med slikami.

Zaradi vsega naštetega se Motion JPEG uporablja predvsem v digitalnih fotoaparatih za zajem videa ter v nekaterih programih za obdelavo videa. Pri tem je vsaka slika kodirana neodvisno ter tako omogoča urejanje brez dekompresije in ponovne kompresije, kar močno pohitri proces urejanja.

Čeprav je JPEG dobro standardiziran, pa Motion JEPG ni bil nikoli standardiziran. Tako ne obstaja uraden dokument, ki bi opisoval format kodiranja Motion JPEG [7].

MPEG-1

Standard MPEG-1 je bil objavljen leta 1992. Zasnovan je bil z namenom doseganja zadovoljive video kakovosti s podatkovno širino 1.5Mbit/s in resolucijo 352x288/240 pik. MPEG-1 je široko razširjen, saj ga je mogoče predvajati na skoraj vseh računalnikih, DVD in VCD predvajalnikih.

Na dizajn standarda je močno vplival standard H.261. MPEG-1 uporablja metode kompresije, ki izrabljajo percepcijo človeškega vida, barvni prostor YCbCr in metodo vzorčenja podatkov 4:2:0 . Omogoča resolucije do 4095x4095 in podatkovno širino do 100Mbit/s. Pri tem uporablja več različnih tipov slik, ki so značilne tudi za kasnejše standarde [10]:

- I-slike (Intra-frame) – I-slika je lahko neodvisno dekodirana saj ne potrebuje podatkov o katerikoli drugi sliki. Praktično gledano je to tako še vedno JPEG slika. Zaradi te lastnosti je preskakovanje po videu mogoče samo na I-slike. Kompresija, s samo I-slikami, je hitra vendar prinese

večjo podatkovno širino (faktor 3). MPEG-1 je ponavadi uporabljal razmak 15-18 slik med I-slikami.

- P-slike (Predicted-frame) – P-slike izrabljajo časovno redundanco, kar pomeni, da se sledeče si slike v videu ne spreminjajo veliko. Shranjuje se samo razliko do predhodne slike (druga P-slika ali I-slika). Vektor gibanja je podan za vsak makroblok. Vsebuje lahko tako I- kot P-makrobloke.

- B-slika (Bidirectional-frame) – B-slika uporablja podatke tako predhodne kot sledeče slike. Dekoder mora zato najprej dekodirati sledečo P- ali I-sliko in šele nato lahko dekodira B-sliko. Posledično je v praksi potreben dvakrat večji pomnilnik. Zaradi večje zapletenosti so B-slike včasih nepodprte v dekoderjih in se le redko uporabljajo.

- D-slike (DC-frame) – D-slike so posebna vrsta slik, ki se v kasnejših standardih ne uporabljajo. V osnovi je to I-slika, kompresirana samo z DC koeficienti (AC koeficient je odstranjen), kar prinese neodvisno sliko nizke kakovosti. Uporabljale so se predvsem za hitro iskanje določenega izseka skozi video vsebine.

Makrobloki standarda MPEG-1 so velikosti 16x16 pik, za kvantizacijo pa se uporabljajo bloki velikosti 8x8 pik. Predikcija gibanja se uporablja nad posameznimi makrobloki.

Pomembna pomanjkljivost standarda je, da omogoča samo progresivno (neprepleteno) predvajanje slik, kar je v nasprotju s televizijskima standardoma PAL in NTSC, ki prepletanje uporabljata. Posledično je z razvojnega vidika standardu hitro sledil njegov naslednik, standard MPEG-2, ki prepleteno predvajanje slik omogoča.

MPEG-2 / H.262

Standard MPEG-2 / H.262 je bil razvit s sodelovanjem organizacij ISO in ITU. S strani obeh standardizacijskih organizacij je bil potrjen leta 1993. Podpira standardno (SD, 720x576/480 pik) in visoko ločljivost (HD 1920x1080 pik) video vsebine.

Zaradi vključitve podpore visoki ločljivosti je bil prekinjen razvoj standarda MPEG-3. Standard MPEG-2 / H.262 je še vedno najbolj razširjen video standard zaradi uporabe na DVD predvajalnikih, široko podporo pa ima tudi pri video kamerah in TV sprejemnikih z digitalnim virom signala (DVB-T/S/C in Ameriški ATSC).

Standard MPEG-2 / H.262 je bil zasnovan za namene studijske kakovosti videa pri širini med 3-10Mbit/s za SD ločljivost. Posledično ni primeren za video podatkovne širine pod 1Mbit/s. Dekoder za stndard MPEG-2 je kompatibilen s standardom MPEG-1 in uporablja vse tehnike zasnovane v standardu MPEG-1 [2].

Digital Video (DV)

Standard DV za video kompresijo je bil razvit s strani organizacije IEC in odobren leta 1994. Oblikovan je bil predvsem za uporabo na video kamerah, ki shranjujejo video podatke na tračne enote.

DV ne uporablja kompenzacije gibanja, namesto tega vsako posamezno sliko zakodira s fiksno podatkovno širino 25Mbit/s. Skupaj z avdio signali in podatki za detekcijo in korekcijo napak to doprinese veliko podatkovno širino – 36Mbit/s. Standard DV uporablja DCT transformacijo in vzorčenje po metodi 4:1:1 ter 4:2:0. Sposobnost DV je boljša kot pri Motion JPEG in je podobna zmogljivosti MPEG-2 / H.262. Pristop brez kompenzacije gibanja (uporaba samo I-slik) je bil izbran zaradi možnosti hitrega urejanja ter posledično omogoča hitrejše predvajanje naprej in nazaj naravnost iz magnetnega traku [7].

H.263

Standard H.263, ki je bil odobren leta 1995, je bil zasnovan s strani ITU in predstavlja velik korak naprej v razvoju video kompresijskih standardov. Dandanes je Standard H.263 najbolj razširjen standard za video konference z nizko podatkovno širino in brezžično mobilno telefonijo.

Razvoj H.263 lahko opišemo kot nadgradnjo H.261 in MPEG-2 / H.262 standardov. H.263 se je izkazal za boljšega od vseh predhodno razvitih standardov pri vseh podatkovnih širinah. Razlika je še posebej vidna pri nepreplетenih video posnetkih. Pri nizkih podatkovnih širinah je kakovost videa bolšja za rang 2 v primerjavi z MPEG-2 / H.262. Standard H.263 ima več verzij in veliko aneksov. Ravno tako je njegov razvoj prepleten z razvojem MPEG-4 standarda, saj so ju delno razvijali vzporedno v istem času.

Standard H.263 se uporablja v standardih za video konference, v uporabi pa je tudi na internetu, saj se uporablja v vtičniku Flash, ki ga podpira na primer YouTube. Standard za mobilne telefone 3GPP vključuje H.263 za prenos videa med mobilnimi telefoni [13].

RealVideo

Standard RealVideo za kompresijo videa zasluži omembo, saj je eden redkih standardov razvitih s strani zasebne družbe, v tem primeru RealNetworks. RealNetworks je bila ena prvih komercialnih družb, ki je prodajala orodja za prenos avdio in video podatkov preko interneta.

Prva enačica njihovega kodeka, je bila razvita leta 1997 in je temeljila na H.263 kodeku. Od enačice 8 naprej je družba začela uporabljati lasten lastniški kodek, ki je baziran na prvem osnutku video standarda H.264 [7].

MPEG-4 (part 2 / SP / ASP)

Standardizacija kodeka MPEG-4 se je začela leta 1995, vendar je bil kodek večkrat izboljšan z novimi profili, ki so vsebovali nove modernejše koncepte kompresije, kot na primer kodiranje objektov in oblik, kompresija slik na osnovi valčkov, modeliranje obrazov, 3D grafika ...

Kljub veliko novostim je le malo novodobnih tehnik našlo pot v komercialne produkte. Kasnejši napori pri standardizaciji, so bili usmerjeni predvsem v kompresijo navadnega videa.

MPEG-4 standard je bil zasnovan z namenom dovoljevanja velikega razpona kakovosti v razmerju s podatkovno širino. Uporablja vrsto profilov, ki definirajo nabor funkcij, ki jih mora določena naprava podpirati. Najpogostejša sta [2]:

- Simple profile standard (SPS) je najosnovnejši profil in je kompatibilen z H.263.
- Advance simple profile (ASP) ima podporo za SD video, prepletanje in dodatna orodja za povečanje kompresijske učinkovitosti. Najbolj poznani izvedbi profila ASP sta gotovo lastniška video standarda DivX / XviD. V začetku tega tisočletja je bil ASP eden izmed najbolj popularnih video standardov, sploh v času, ko so bili postopki kompresije in dekompresije predani v javno domeno.

V svojem bistvu MPEG-4 uporablja enake tehnike kodiranja kot standard MPEG-2.

VP3 - VP7

Video standardi, razviti s strani On2 Tehnologies, so poznani pod imeni VP3 do VP7. Leta 2004 je Adobe izbral standard VP6 za uporabo v svojem video

standardu Flash 8. Flash je pogosto uporabljena tehnologija za prikaz videa na internetu, uporablja jo npr. YouTube, pa tudi druge internetne strani. Ravno tako je Skype uporabil verzijo standarda On2 za video konference preko protokola IP.

Leta 2010 je družbo On2 Tehnologies kupila družba Google, ki je kasnejše verzije te družine standardov ponudila pod prosto licenco BSD [7].

WMV9 / VC-1

Microsoft je razvil standard za kompresijo videa Windows Media Video version 9, in sicer najprej kot lastniški standard, nato pa ga je leta 2006 standardizirala organizacija SMTPE.

Standard je hibriden in uporablja konvencionalno kompenzacijo gibanja, DCT transformacijo in Huffmanovo kodiranje, podobno kot H.263 in MPEG-4. Glavna razlika je v uporabi blokov velikosti 4x4 in ne 8x8, kar najdemo v prejšnjih dveh standardih.

Standard naj bi bil podoben ITU standardom H.264 / MPEG-4 AVC, vendar so študije pokazale, da je manj zmogljiv. Ker pa je računsko manj zahteven kot H.264 / MPEG-4 AVC, je bil podprt s strani HD DVD in Blu-Ray diskov kot obvezni standard za predvajalnike in opcijski standard za proizvajalce diskov. Kodek se uporablja tudi na internetu in v igralni konzoli Xbox 360.

Proti standardu WMV9 / VC-1 je bilo vloženih veliko patentnih zahtevkov, kar ga razlikuje in podraži v primerjavi z drugimi lastniškimi kodeki, kot na primer kodeki podjetja On2 Tehnologies [7].

Theora

Theora je video standard, podan s prostimi licencami, osnovan iz standarda VP3 družbe On2 Tehnologies. Prvi osnutek je bil podan leta 2002, prva stabilna verzija pa je bila objavljena leta 2008.

Po kakovosti se lahko primerja s standardom MPEG-4, najnovejša enačica pa se lahko po testih primerja tudi z standardom H.264 / AVC. Uporablja bloke velikosti 8x8, ki so nato združeni v superbloke velikosti 4x4, DCT transform in Huffmanovo kodiranje entropije [15].

1.1.3 Trenutno stanje

H.264 / MPEG-4 part 10 / AVC

Medtem ko je odbor za standardizacijo MPEG-4 posvetil veliko časa modernim metodam kompresije, je H.264 začel z osnovami. Cilj standarda je bil kompresija pri dvakrat manjši podatkovni širini z enako kakovostjo slike. V začetku se je standard imenoval H.26L oziroma JVT (po skupni ekipi v kateri sta sodelovali organizaciji ISO in ITU), organizacija ITU je zanj uporabljala ime H.264, organizacija ISO pa ime MPEG-4 part 10 / AVC.

Standard uporablja več zmožnosti za dosego svojega cilja:

- Uporaba predikcije med slikami ni več omejena samo na predhodno in sledečo (B-slike), ampak se pri tem lahko uporablja do 16 referenčnih slik.
- Variabilna velikost bloka za predikcijo gibanja, ki ima razpon med 4x4 do 16x16 pik.
- Zmožnost uporabe vseh tipov makroblokov znotraj B-slike.
- Zmožnost kompenzacije slike z večjo natančnostjo kot slikovna pika.
- Prostorska predikcija z robov sosednjih makroblokov.
- Napredno brezizgubno kodiranje makroblokov.
- Filter za popravljanje napak zaradi uporabe makroblokov.
- Več načinov kodiranja entropije (CABAC, CVLC, VLC).
- ...

Zaradi velike izboljšave kvalitete je H.264 hitro postal vodeči standard in je bil kmalu uporabljen v video aplikacijah na napravah iPod kot tudi v TV standardih oddajanja DVB-H in DMB. Manj zmogljive naprave pri tem uporabljajo osnoven Baseline profil, medtem ko lahko bolj napredne naprave uporabljajo Main in High profil pri ločljivosti HD. Baseline profil ne podpira prepletenega predvajanja slik, medtem ko ga višji profili podpirajo [16].

AVS

Vlada Kitajske je sprožila izdelavo lastnega avdio video standarda (AVS), ki je bil dokončan leta 2005. Eden izmed pomembnejših faktorjev pri izdelavi je bil omogočanje neodvisnosti od zahodnih standardov in patentov.

Standard AVS je po uspešnosti kodiranja primerljiv s standardom H.264, medtem ko je računska zahtevnost manjša.

Kodek je uporabljen v sistemih TV in IPTV znotraj Kitajske. Zunaj nje ni nikoli dosegel splošne uporabe, saj so se pojavili drugi standardi, ki so prosto dostopni [7].

VP8

Video standard VP8 je bil objavljen leta 2008 s strani družbe On2 Tehnologies. Ko je to družbo leta 2010 kupila družba Google, je le-ta kasneje standard objavila kot prosto dostopen pod odprto kodno licenco.

Glavne tehnike, ki jih standard VP8 uporablja, so:

- Blok za kvantizacijo v velikosti 4x4 pik.
- Koeficienti iz 16x16 so lahko dodatno preračunani s pomočjo Walsh-Hadamard transformacije.
- Za predikcijo gibanja objektov med slikami uporablja do tri referenčne slike.
- Uporablja še dodatne metode predikcije gibanja objektov med slikami in v sliki.
- Natančna interpolacija za predikcijo gibanja do osmine pike.
- Adaptiven filter za odstranjevanje napak zaradi blokov.
- Kodiranje entropije na nivoju posamezne slike.
- ...

Testi so pokazali, da je učinkovitost standarda VP8 praktično enaka učinkovitosti standardu H.264. Zaradi proste dostopnosti in učinkovitosti je standard VP8 postal de-facto standard na internetu in ga podpirajo vsi večji brskalniki [3].

1.1.4 Razvoj

H.265 / HEVC

Standard H.265 / HEVC je naslednik standarda H.264 / AVC, ki sta ga skupaj razvili organizaciji ISO in ITU. Standard H.265 / HEVC je bil objavljen leta 2013, njegov cilj pa je bil ponovno prepoloviti količino podatkov, potrebnih za enako kakovost videa kot pri standardu H.264 / AVC.

Standard podpira ločljivost do 8192x4320 in uporablja enako hibridno osnovno strukturo kot vsi standardi od H.261 naprej. Njegove lastnosti, ki so posebno zanimive:

- Zasnovan je tako, da ne vsebuje posebnih metod za kodiranje prepletene slike. Namesto tega se le-ta obravnava kot vsaka druga progresivna slika z določenimi lastnostmi, ki jih kodek prebere iz meta podatkov.
- Namesto makroblokov uporablja enote Coding Tree Unit (CTU), ki so lahko sestavljene iz večjih blokov do velikost 64x64, kar lahko bolje razdeli sliko v bloke.
- Podpira razdelke (Tiles), ki so med seboj neodvisni in tako omogočajo paralelno procesiranje.
- Za kodiranje entropije uporablja algoritem CABAC, ki je podoben algoritmu CABAC iz predhodnega standarda.
- Specificira 33 možnosti intra-predikcije v primerjavi z 8 iz H.264 / AVC.
- Ima izboljšano kompenzacijo gibanja z 8-točkovno interpolacijo.
- Ima dva filtra za popravljanje napak dekodiranja.

Dodatne zmogljivosti, ki še dodatno izboljšajo kompresijo, so bile nato dodane v razširitvah [14].

VP9

Standard VP9 je naslednik standarda VP8 in je bil prvič objavljen leta 2013.

Prinaša več izboljšav v primerjavi z VP8, največja pa je zagotovo podpora za superbloke velikosti 64x64, ki skupaj s posebno drevesno strukturo (quadtree) omogoča boljšo kompresijo [8]. Druge izboljšave so še:

- Dodatni modeli predikcije pri intra- in inter-predikciji.

- Interpolacija s tremi različnimi načini in 8-točkovnim filtrom.
- Tri vrste transformacije: DCT, asimetrični diskretni sinusni transform (ADST) in Walsh-Hadamard Transform (WHT).
- Aritmetično kodiranje entropije.
- Filter za popravljanje napak blokov, ki zmore popravljati superbloke.
- Segmentacija – omogoča razdelitev slike na do 8 delov, katerim lahko dodelimo določene lastnosti (absolutno vrednost, delto, moč filtra ...).
- Razdelki (Tiles) za paralelno procesiranje.

Rezultati testov kažejo, da je VP9 konkurenčen standardu H.265 / HEVC, zaradi prosto dostopne licence pa lahko pričakujemo široko podprtost [6].

1.2 Kodirnik Daala

Video standard Daala je naslednik standarda Theora in je prosto dostopen. Prvi osnutek je bil objavljen leta 2010, vendar je še v razvoju, tako da končna verzija standarda še ne obstaja. Zaradi naprednih in nekonvencionalnih pristopov h kodiranju velja za pomemben preizkusni korak pri razvoju video kompresijskih standardov [19].

Glavna lastnost, ki standard loči od ostalih, je, da ne uporablja navadnega DCT transforma za prehod v frekvenčno domeno, ampak uporablja dve drugi metodi:

- obliko DCT s prekrivanjem blokov, kar odstrani težave z napakami, ki se pojavijo na robovih blokov,
- Haarovo preslikavo.

Za intra- in inter-predikcijo uporablja metode podobne standardu H.264. Kodiranje entropije je izvedeno z različico aritmetičnega kodiranja (Range encoding), ki deluje tudi na nebinarnih podatkih, kar omogoča hitrejše kodiranje v primerjavi z aritmetičnim.

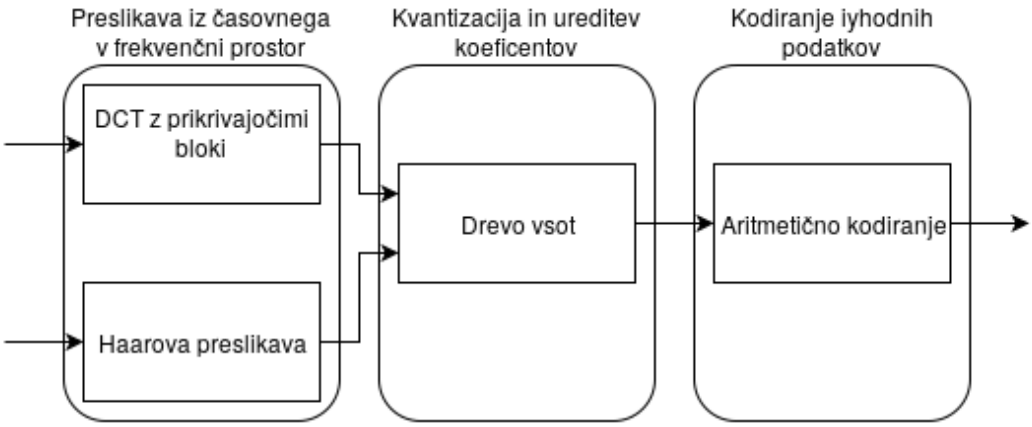
Testi kažejo, da je v določenih primerih standard Daala že enakovreden, če ne že boljši od svojih konkurentov [9].

Standard Daala je bil predlagan kot kandidat za video standard NETVC, ki naj bi poenotil uporabo vrste video vsebin na internetu. Če bo tvegan poskus razvoja uspel, lahko postane standard Daala vodilni standard na področju video kompresije v naslednjih 5 do 10 letih.

1.2.1 Algoritmi

Osnovna zgradba kodirnika je preprosta in podobna vsem prej naštetim kodirnikom. Ima tri glavne gradnike, njihov preprost diagram pa lahko vidimo na sliki 1.1:

- preslikavo iz časovnega v frekvenčno območje,
- obdelavo in razporejanje tako pridobljenih koeficientov slike
- kodiranje podatkov (za čim manjšo velikost izhodnih podatkov).



Slika 1.1: Osnovni gradniki cevovoda

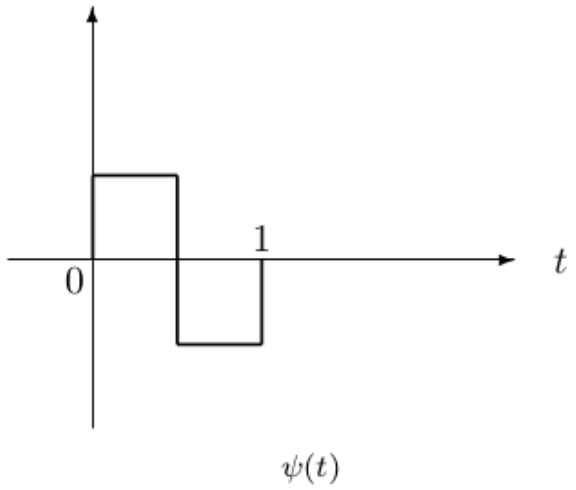
Od predhodnih kodekov se razlikuje po tem, da uporablja nekoliko drugačne algoritme. Za preslikavo uporablja Haarovo valčno preslikavo in DCT s prekrivajočimi se bloki, za obdelavo koeficientov metodo drevesa vsot in za kodiranje svojo izvedbo aritmetičnega kodiranja [19].

Haarova preslikava

Haarova preslikava je ena najpreprostejših valčnih preslikav. Preslikava navzkrižno množi dano funkcijo s Haarovim valčkom, podobno kot to počne Fourierjeva transformacija s sinusno funkcijo [12].

Enačba Haarovega valčka:

$$\psi(t) = \begin{cases} 1 & \text{če } 0 \leq t < \frac{1}{2} \\ -1 & \text{če } \frac{1}{2} \leq t < 1 \\ 0 & \text{drugače} \end{cases} \tag{1.1}$$



Slika 1.2: Haarov valček

Haarov valček je preproste obilke - slika 1.2. Iz njega sledi, da je preslikovalna matrika velikosti 2

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Le-ta je nato uporabljena v več ciklih za vsako od podvelikosti supreblokov (2, 4, 8, 16, 32, 64).

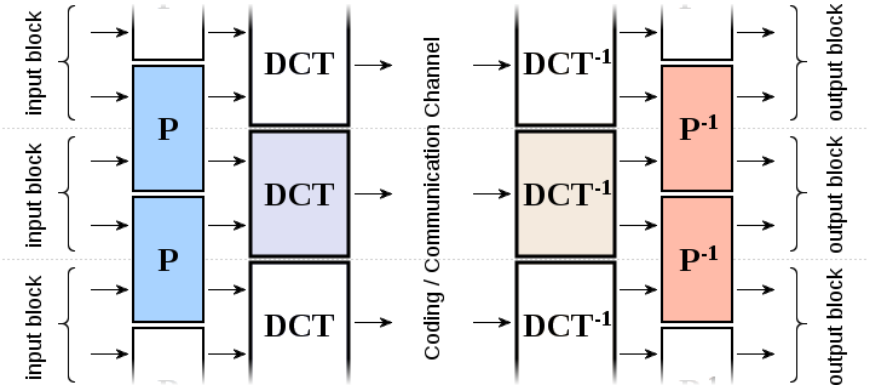
DCT s prikrivajočimi se bloki

DCT s prekrivajočimi se bloki je opredeljena kot DCT preslikava, pri kateri pred njeno uporabo zaženemo popolnoma reverzibilen filter, ki poudari meje blokov [19]. Enako pri dekodiranju slike po IDCT izvedemo še inverzni filter, ki povrne vsebino blokov v prvotno stanje. Filtra se izvajata na dveh sosednjih blokih, diagram delovanja pa prikazuje slika 1.3.

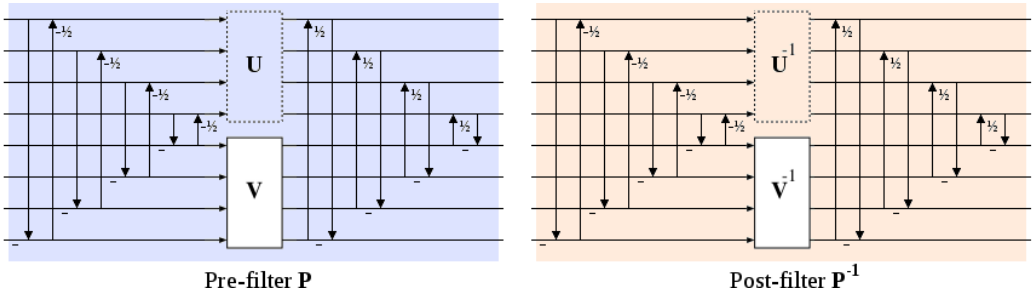
Reverzibilni filter združuje informacije o sosednjih blokih ter tako preprečuje napako na robovih blokov, ki se pojavi pri standardnem DCT. Zgradba filtrov je prikazana na sliki 1.4.

Predikcija koeficenta DC

Za predikcijo DC koeficienta se ne uporablja DC koeficent iz prejšnega bloka, kot v preprostih cevovodih ampak več DC koeficentov iz predhodno izračunanih



Slika 1.3: DCT s prekrivajočimi bloki [19].



Slika 1.4: Predfilter in pofilter, kot jih uporablja kodek Daala, implementirana za 8 točkovno preslikavo DCT. Izkaže se, da je matrika U enaka matriki V, zato jo lahko v implementaciji izpustimo [19].

blokov po sledečih pravilih [18]:
Če z by, bx označimo koordinati bloka v sliki, z Nbx število blokov v x smeri, z $dc_{bx,by}$ DC koeficent tega bloka in $dc\_pred_{bx,by}$ predikcijo tega bloka potem velja:

$$dc\_pred_{bx,by} = \begin{cases} (22 * dc_{by,bx-1} - 9 * dc_{by-1,bx-1} + & \\ 15 * dc_{by-1,bx} + 4 * dc_{by-1,bx+1} + 16)/32 & \text{če } by > 0, bx < Nbx \\ (23 * dc_{by,bx-1} - 10 * dc_{by-1,bx-1} + & \\ 19 * dc_{by-1,bx} + 16)/32 & \text{če } by > 0, bx > 0 \\ dc_{by-1,bx} & \text{če } by > 0 \\ dc_{by,bx-1} & \text{če } bx > 0 \\ 0 & \text{drugače} \end{cases} \tag{1.2}$$

Drevo vsot

Za razvrstitev po kvantizaciji uporablja kodirnik Daala drevesno strukturo vsot [19]. To je rekurzivno zgrajeno drevo za čigar elemente velja:

$$ts_{y,x} = \begin{cases} ts_{2y,2x} + ts_{2y,2x+1} + ts_{2y+1,x} + \\ ts_{2y+1,2z+x+1} + c_{x,y} & \text{če } y < n/2, x < n/2 \\ c_{x,y} & \text{drugače} \end{cases} \quad (1.3)$$

kjer sta x, y indeksa v matriki lokacij, n predstavlja velikost matrike, $c_{x,y}$ koeficent in $ts_{x,y}$ vsoto drevesa. Nato sledi kodiranje tako pridobljenih vsot drevesa.

Aritmetično kodiranje

Za kodiranje podatkov kodirnik Daala uporablja posebno različico aritmetičnega kodiranja, poimenovano tudi kodiranje obsega (Range encoding) [19].

Za razliko od navadnega aritmetičnega kodiranja, ki deluje nad biti, lahko to kodiranje deluje nad črkami abecede z več kot dvema elementoma. Ta vrsta kodiranja je uporabljena tudi zaradi večih patentov, ki obstajajo za navadno aritmetično kodiranje. Ravno tako je kodiranje z več črkami v abecedi hitrejše, saj lahko v enem koraku zakodiramo večjo količino informacij. Daala uporablja kodiranje obsega z abecedami do 16 znakov.

Kodiranje poteka tako, da izberemo dve spremenljivki, ki predstavljata meje obsega - najnižjo vrednost (low) in najvišjo vrednost (high). Nadalje ta obseg razdelimo na toliko obsegov, kot je črk v abecedi. Velikost obsega je odvisna od verjetnosti za dano črko. Če so vse črke enako verjetne, potem so posledično vsi obsegi enaki.

Ko prejmemo črko abecede, ki jo želimo kodirati, se obseg, ki je na voljo, zamenja z obsegom dane črke. Novi obseg ponovno razdelimo med črke abecede.

Tako postopoma manjšamo velikost obsega. Po več ponovitvah lahko ugotovimo, da se določeni biti obsega ne spreminjajo več. Te bite lahko shranimo, saj predstavljajo izhodne podatke, nato pa obseg spet povečamo na večjo vrednost.

Primer:
Imamo tri črke A, B in C z verjetnostjo (A: 0.60; B: 0.20; C: 0.20) ter obseg od 0 do 100000. Kodiramo zaporedje AABAC
low=0
high=10000

A: [0, 60000]
B: [60000, 80000]
C: [80000, 100000]

Prejmemo črko A, nov obseg prevzame vrednosti A, ki so od 0 do 60000. Ta korak je grafično prikazan na sliki 1.5.
low=0
high=60000
AA: [0, 36000]
AB: [36000, 48000]
AC: [48000, 60000]

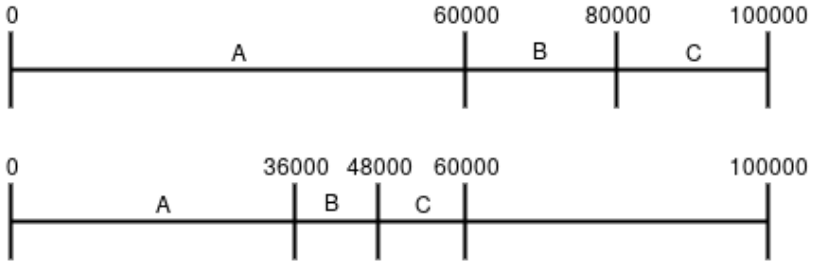
Ponovno prejmemo črko A, nov obseg prevzame vrednosti A, ki so od 0 do 36000.
low=0
high=36000
AAA: [0, 21600]
AAB: [21600, 28800]
AAC: [28800, 36000]

Prejmemo črko B, nov obseg prevzame vrednosti B, ki so od 21600 do 28800.
low=21600
high=28800
AABA: [21600, 25920]
AABB: [25920, 27360]
AABC: [27360, 28800]

Ponovno prejmemo črko A, nov obseg prevzame vrednosti A, ki so od 21600 do 24192.
low=21600
high=24192
AABAA: [21600, 24192]
AABAB: [24192, 25056]
AABAC: [25056, 25920]

Prejmemo črko C; naš končni obseg in rezultat je katerakoli vrednost z obsega C, katerega meje so so od 25056 do 25920.

Opazimo lahko, da sta prvi dve števki (25) mejnih vrednosti obsega enaki. Za nadaljnje kodiranje bi jih lahko shranili (poslali na izhod), nove vrednosti



Slika 1.5: Prikaz zmanjševanja obsega pri prvem koraku primera aritmetičnega kodiranja - vhodna črka A.

obsega pa nato pridobimo z zamikom preostalih števk, za toliko mest kot smo jih shranili. V našem primeru bi bile nove vrednosti:
low=5600
hig=92000

1.3 Namen magistrskega dela

Namen tega dela je preučiti zmožnost implementacije osnovnega cevovoda video kodirnika Daala ter implementacija takšnega cevovoda. Osnovni cevovod predstavlja cevovod za implementacijo I slik brez predikcije med slikami in brez naprednih funkcij, kot na primer optimizacija RDO. Cevovod naj bi podpiral velikost slik do polne visoke ločljivosti 1920x1080 in hitrost ene slikovne pike na urin cikel.

Poglavje 2

Implementacija

2.1 Razvojno okolje

2.1.1 Strojna oprema

Za implementacijo kodirnika je bila izbrana razvojna plošča PCIE s čipom Xilinx FPGA XC7K410T iz družine Kintex-7, saj je dovolj velika in omogoča testiranje preko PCIE vodila [17].

FPGA čip ima sledeče osnovne podatke:

- Število celic: 406720
- Število DSP rezin: 1540
- Velikost pomnilnika: 795 (32Kb)
- Število I/O mest: 500

2.1.2 Programska oprema

Implementacija je bila pripravljena v operacijskem sistemu GNU/Linux, različica Fedora 24. Za delo se je uporabljalo uradno razvojno orodje za čipe Xilinx FPGA, razvojno okolje Vivado, ki omogoča vse korake razvoja - urejanje izvorne kode, simulacijo le-te in na koncu tudi njeno sintezo in implementacijo [17]. Za razvoj same izvorne kode je bil uporabljen urejevalnik besedil VIM.

Izvorna koda

Izvorna koda implementacije je spisana v jeziku za opis digitalnih vezji in verifikacijo SystemVerilog, ki se je prvič pojavil leta 2002. Je nadgradnja jezika za opis digitalnih vezji Verilog iz leta 1984. Zadnja različica jezika je bila izdana leta 2013. Jezik omogoča tako pisanje kode namenjene implementaciji na čipu, kot tudi pisanje testnih modelov z višje nivojskimi funkcijami, objektnim programiranjem in razvoj celotnih testnih okolij, kot sta OVM in UVM.

2.1.3 Testno okolje

Testiranje je bilo izvedeno v dveh korakih. Prvi korak je predstavljal testiranje implementacije v simulacijskem okolju, kjer so bili posamezni moduli cevovoda testirani do stopnje, ko so pravilno delovali na manjši množici vhodnih podatkov. Tako so bile testirane vse glavne komponente in povezovalna logika med njimi.

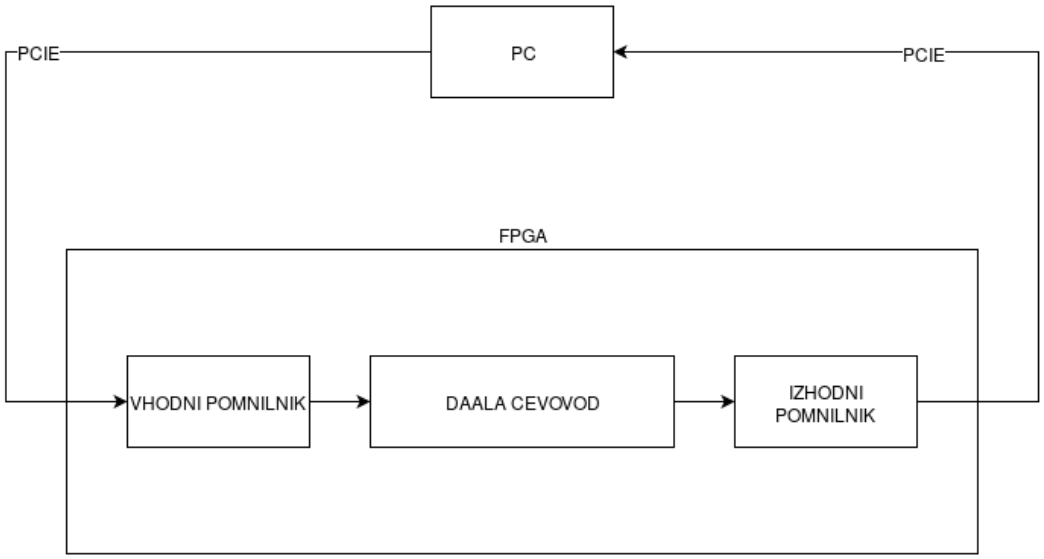
Simulacija se je izkazala za prepočasno za testiranje celotnega cevovoda, saj bi simulacija slike pri normalni ločljivosti trajala več ur. Zato se je nadaljnje testiranje izvajalo neposredno na FPGA čipu. Vhodni podatki so se preko vodila PCIE prenesli v pomnilnik na čipu FPGA. Pomnilnik je bil nato priključen na vhodno vodilo Daala cevovoda. Izhod iz cevovoda je bil ponovno priključen na pomnilnik, čigar podatki so se preko vodila PCIE prebrali nazaj v testni računalnik. Podatki so se nato primerjali s podatki pridobljenimi z referenčno implementacijo cevovoda v programskem jeziku C. Diagram testnega okolja je prikazan na sliki 2.1.

2.2 Problemi pri implementaciji

Med samim načrtovanjem implementacije in raziskovanjem zmožnosti referenčne implementacije so se pokazale nekatere težave, ki so omejile razvoj cevovoda [18].

2.2.1 Nedelovanje razbitja superbloka na bloke

Največja težava za izvedbo implementacije predstavlja samo stanje referenčne implementacije [18], ki ni polno delujoče. Cilj projekta je bil priprava osnovnega cevovoda brez naprednih funkcij. Ena izmed naprednih funkcij, ki jo kodek Daala uporablja je metoda RDO optimizacije glede na stopnjo popačenja slike. Ta metoda omogoča prilagajanje izgube podatkov glede na samo kvaliteto



Slika 2.1: Testno okolje

tako pridobljene izhodne slike. Ker lahko predvidevamo, da testni vektorji za referenčno implementacijo RDO optimizacijo uporabljajo, referenčni model, kljub možnosti za njeno izključitev, brez nje ne deluje.

Napaka se pojavi pri razbitju superbloka na podbloke, ki v načinu brez RDO ne deluje. Posledično ni mogoče uporabljati preslikave DCT s prekrivajočimi bloki. Deluje le Haarova valčna preslikava, ki pa je implementirana le za delovanje v brezizgubnem načinu, kar posledično pomeni, da kvantizacije ni mogoče uporabljati. Haarova valčna preslikava brez kvantizacije je tako edina delujoča množica nastavitev, ki deluje brez funkcije RDO, zato je bila v našem primeru uporabljena v implementaciji.

Opisana napaka je poznana že dalj časa vendar zaenkrat še ni bila odstranjena [5].

2.2.2 Zahteve po pomnjenju celotnega superbloka

Algoritmi, ki se v cevovodu uporabljajo, imajo zahtevo po pomnjenju celotnega bloka, ki je v največjem primeru kar velikost superbloka. Haarova preslikava ima namreč več prehodov nad istimi podatki in za vsak prehod potrebuje rezultate prejšnjih prehodov. Ravno tako algoritem za izračun drevesa vsot zaključi z obdelavo bloka na DC koeficientu, ki pa je prvi, ki je potreben za prehod v aritmetično kodiranje. Kot Haarova preslikava tudi algoritem za izračun drevesa vsot in algoritem za kodiranje le-tega potrebujeta vse podatke

o koeficientih.

Zaradi vseh naštetih lastnosti je bil opuščen standardni način prehoda podatkov, kjer si jih posamezni moduli med seboj podajajo preko vodila. Namesto tega sta bili uporabljeni dve skupini bločnih pomnilnikov – skupina za izračunane koeficiente ter skupina za izračunano drevo vsot. Moduli imajo dostop do omenjenih pomnilniških skupin in jih uporabljajo za shranjevanje in nadaljnjo računanje.

2.2.3 Zapletena izvedba aritmetičnega kodiranja

V želji po čim manjši izhodni velikosti podatkov se za kodiranje izhodnih podatkov uporablja posebna izvedba aritmetičnega kodiranja, ki pa je bistveno bolj zapletena od Huffmanovega kodiranja, ki ga poznamo iz starejših kodirnikov.

Izvedba v kodirniku Daala vsebuje več poti kodiranja glede na vrsto vhodnega podatka, saj končna oblika kodiranja še ni določena, zato je tudi njena izvedba bolj zapletena.

Ravno tako predstavlja kodiranje podatkov ozko grlo samega cevovoda, saj potrebujemo trenutno stanje notranjih spremenljivk, ki se spremenijo ob kodiranju trenutnega podatka za kodiranje naslednjega.

2.2.4 Računske operacije

Referenčni model kodirnika je spisan z mislijo na strojno implementacijo [18]. Ne uporabljajo se napredne matematične funkcije, ki jih je težje implementirati v strojni opremi.

V kodi je nekaj operacij množenja in samo ena operacija deljenja nad dvema pravima spremenljivkama.

2.3 Zgradba implementacije

Implementacijo sestavlja 23 modulov in 5 IP gradnikovn, generiranih z razvojnim okoljem Vivado. Diagram povezav prikazuje slika 2.2, seznam modulov s kratkim opisom pa tabela 2.1.

Vhodno vodilo je priključeno na modul Reorder, ki vsebuje pomnilnik za 64 vrstic slike. Podatke je potrebno preurediti iz vrstic v superbloke velikosti 64x64 pik, nad katerimi se izvaja nadaljnja obdelava.

Modul Reorder je nato preko povezovalne logike priključen na modul Haar, v katerem se izvede Haarova preslikava iz časovne v frekvenčno domeno.

Tabela 2.1: Seznam uporabljenih modulov

Ime modula	**Opis**
daala_pkg	Definicije parametrov, konstant in večkrat uporabljenih funkcij
daala_top	Glavni modul, ki zajema vse preostale module in vhodno ter izhodno vodilo
reorder	Spreminjanje toka podatkov iz zaporedja vrstic na bloke velikosti 64x64 pik
haar_switch	Izbira med predpomnilniki barv Y,U in V, ki se procesirajo ločeno
haar	Logika za izvedbo Haarove preslikave
haar_kernel	Jedro Haarove preslikave
haar_dc_sb	Predikcija DC koeficenta med bloki
max_tree	Izračun drevesa vsot
max_tree_entenc	Logika za izvedbo kodiranja drevesa vsot
gen_enc	Aritmetično kodiranje DC koeficenta
laplace_enc	Kodiranje, uporabljeno pri kodiranju DC koeficenta
sb_buff	Skupina 3 bločnih pomnilnikov, kjer se hranijo koeficienti bloka v posameznem koraku
sb_tree_sum_buff	Skupina 2 bločnih pomnilnikov, kjer se hranijo podatki drevesa vsot
entenc_buff	Pomnilnik izhodnih podatkov
entenc_calc	Izračun novega obsega pri aritmetičnem kodiranju
entenc_cdf_adapt	Priprava aritmetičnega kodiranja z adaptivno tabelo in adaptiranje tabele
entenc_cdf_adapt_switch	Izbira vhoda v aritmetično kodiranje z adaptivno tabelo
entenc_ctrl	Nadzor aritmetičnega kodiranja in priprava glave posamezne slike
entenc_norm	Normiranje obsega pri aritmetičnem kodiranju in shranjevanje stanja
entenc_uint	Priprava aritmetičnega kodiranja pozitivnega celega števila
entenc_bits_switch	Izbira vhoda za aritmetično kodiranje poljubnega števila bitov
entenc_bits	Priprava aritmetičnega kodiranja poljubnega števila bitov
entenc_done	Zaključevanje aritmetičnega kodiranja

Tabela 2.2: Signali AXI4-Stream vodila

Ime signala	**Opis**
TVALID	Veljavni podatki
TREADY	Pripravljen na prejemanje podatkov
TDATA	Podatki
TLAST	Zadnji podatek

Po preslikavi se izvede predikcija DC koeficienta v modulu Haar_dc_sb. Nato se v modulu Max_tree izračuna drevo vsot.

Pripravljeni podatki se sproti pošiljajo algoritmu za aritmetično kodiranje. Glava izhodnih podatkov se ustvari v modulu Entenc_ctrl, nadalje se aritmetičnemu kodiranju poda DC koeficient in nazadnje še ostali koeficienti skupaj z drevesom vsot.

Po končani obdelavi celotne slike se sproži še končni algoritem aritmetičnega kodiranja, ki shrani preostale podatke in jih pripravi za izhod.

2.3.1 Uporabljeni IP gradniki

V implementaciji je uporabljenih 5 IP generiranih gradnikov, ki omogočajo lažje delo z FPGA čipom.

Pripravljene so bile tri velikosti bločnega pomnilnika RAM. Vse tri imajo dve vhodno/izhodni vodili:

- pomnilnik vrstic: 122880x8 (64 vrstic po 1920 8 bitnih pik, kar je potrebno za obdelavo videa polne visoke ločljivosti),
- pomnilnik podatkov superbloka: 4096x16 (blok 64x64 s 16-bitnimi podatki),
- pomnilnik podatkov superbloka: 2048x32 (blok 64x64 s 16-bitnimi podatki, kjer shranimo 2 podatka hkrati).

Ravno tako je bil generiran ROM velikosti 128x256 za tabelo koeficientov, ki se uporabljajo pri aritmetičnemu kodiranju.

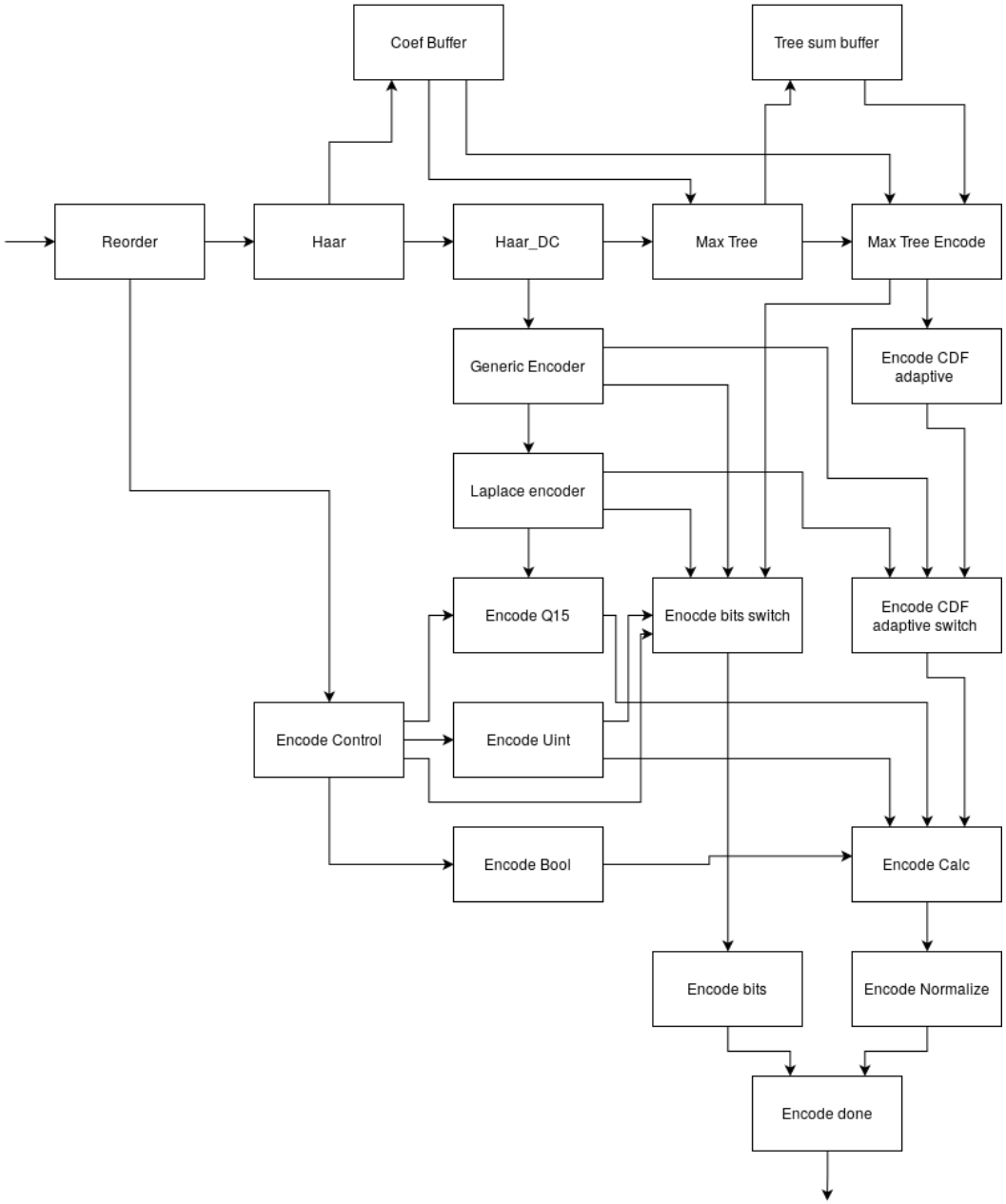
Nenazadnje je bil generiran tudi delilnik, ki je potreben za delovanje generičnega kodiranja (ene izmed metod, ki jih podpira aritmetično kodiranje).

2.3.2 VODILO AXI4-Stream

Za povezovanje modulov se uporablja vodilo, osnovano na standardu AXI4-Stream. Vodilo AXI4-Stream je del družine vodil mikrokontrolerjev ARM

AMBA, prvič predstavljenih v letu 1996. Verzija AMBA 4.0, objavljena leta 2010, vsebuje drugo večjo revizijo vodil AXI - revizijo AXI4 [1].

Signali vodila AXI4-Stream so prikazani v tabeli 2.2. Vodilo ima prednost predvsem v tem, da je de-facto industrijski standard, ki ga podpirajo vsi proizvajalci. Prav tako je dobro prilagodljivo, saj omogoča raznovrstno uporabo znotraj svojih okvirov.



Slika 2.2: Diagram zgradbe cevovoda

Poglavje 3

Rezultati

3.1 Zahtevnost algoritmov

3.1.1 Haarova preslikava

Jedro Haarove preslikave je zelo preprosto, saj ne vsebuje množilnikov in delilnikov. Sestavljeno je iz 7 seštevalnikov in enega aritmetičnega pomika v desno. V funkcijo vstopajo 4 spremenljivke, ravno toliko je tudi izhodnih podatkov.

Celotna Haarova preslikava se izvede v več krogih, vsakokrat nad za četrtino manjšem številu podatkov. Iz tega lahko izračunamo, da je časovna zahtevnost celotne preslikave:

$$O\left(\frac{1}{3}n^2\right) \tag{3.1}$$

Ker za pomnjenje rezultatov uporabljamo pomnilnik RAM z dvema vodiloma in je pri tem potrebno prebrati štiri vhodne podatke ter shraniti en izhodni podatek, sta za vsak izračun potrebna 2 urina cikla. Iz tega sledi, da modul efektivno deluje s polovično hitrostjo.

3.1.2 Drevo vsot

Drevo vsot je rekurzivni algoritem, ki potuje od korena drevesa (0,0) do listov. V FPGA modulu je implementiran kot iterativni algoritem, ki začne v skrajnem listu (63,63) in potuje v obratnem vrstnem redu po celotnem bloku. Časovna zahtevnost je torej

$$O\left(n^2\right) \tag{3.2}$$

Operacije samega algoritma so preproste - listom drevesa se priredi vrednost koeficienta, medtem ko se vejam priredi seštevek 4 potomcev in koeficienta. Potrebni so torej 4 seštevalniki.

3.1.3 Aritmetično kodiranje

Aritmetično kodiranje je zaradi nedokončanosti algoritma zapleteno, saj uporablja 6 različnih poti obdelave podatkov. Najpreprostejša pot, ki zakodira 1 bit, zahteva 1 množilnik in 2 seštevalnika, medtem ko adaptivni algoritem z abecedo velikosti 16 zahteva 9 seštevalnikov in 1 množilnik ter dodatno logiko za izbiro in adaptacijo razpona obsegov posamezne črke po kodiranju.

Vse vrste kodiranja se združijo pri normiranju končnega obsega. To predstavlja že omenjen postopek shranjevanja že izračunanih bitov in s tem ponovnega povečevanja obsega.

Sama zahtevnost algoritma je majhna, povečuje se linearno z rastjo podatkov:

$$O(n) \tag{3.3}$$

3.2 Prostorska zahtevnost

Celoten cevovod z vsemi pomnilniki se je zlahka umestil na čip FPGA. Na sliki 3.1 so podani podatki za celoten cevovod - modul daala_top in vse podmodule.

Cevovod porabi 62546 rezin LUT, kar predstavlja 25% vseh rezin na čipu FPGA. Največ jih porabi modul za adaptivno kodiranje, saj vsebuje 128 tabel za obsege črk abecede.

Prav tako celoten cevovod zasede 135797 registrskih rezin, kar je 30% vseh registrskih rezin, ki jih je v čipu FPGA na voljo. Tudi v tem primeru je največji porabnik prav modul za adaptivno kodiranje, ponovno zaradi velikega števila adaptivnih tabel.

Uporabljenih je 143 32Kb bločnih pomnilnikov, kar je 18% vseh bločnih pomnilnikov. Pri tem jih največ porabi modul Reorder - kar 96 32KB blokov, saj vsebuje pomnilnik 64 vhodnih vrstic.

3.3 Hitrost delovanja

Celotna logika cevovoda, vključno s pomnilniki, teče pri frekvenci ure 125MHz. Vhod v cevovod je širine 1 pike (3 barve) na urin cikel. Dva izmed modulov

Name	Slice LUTs (254200)	Slice Registers ...	F7 Muxes (127100)	F8 Muxes (63550)	Slice (63550)	LUT as Logic (2...	LUT as Memo...	LUT Flip Flop Pair...	Block RAM Tile (795)
daala_top_i (da...	62546	153797	18420	7539	43954	62546	0	1209	143
entenc_bits_i ...	60	71	0	0	44	60	0	15	0
entenc_bits_s...	90	37	0	0	60	90	0	0	0
entenc_buff...	69	4	18	9	31	69	0	0	30
entenc_calc...	174	49	0	0	87	174	0	1	0
entenc_cdf_a...	31425	108321	14370	6144	30294	31425	0	98	0
entenc_cdf_a...	264	49	0	0	98	264	0	0	0
entenc_ctrl_i ...	92	23	0	0	42	92	0	17	0
entenc_done...	395	211	0	0	185	395	0	151	0
entenc_norm...	272	119	0	0	157	272	0	25	0
entenc_uint...	46	3	0	0	34	46	0	0	0.5
gen_enc_i (g...	637	595	42	0	399	637	0	111	0.5
haar_dc_sb_i ...	22958	43044	3864	1359	12897	22958	0	29	0
haar_i (haar)	3353	833	0	0	2350	3353	0	87	2
haar_switch...	24	12	0	0	11	24	0	6	0
laplace_enc...	466	155	97	27	194	466	0	96	4
max_tree_ent...	1470	103	28	0	947	1470	0	59	0
max_tree_i (m...	82	61	0	0	58	82	0	32	0
reorder_i (reo...	213	90	1	0	102	213	0	57	96
sb_buff_i (sb_...	144	14	0	0	80	144	0	2	6
sb_tree_sum_...	342	3	0	0	139	342	0	0	4

Slika 3.1: Rezultati implementacije FPGA

(Haar in max_tree) pa potrebujeta za svoje delovanje 2 urina cikla na barvo slikovne pike. Tako je idealen pretok 6x manjši in torej znaša:

$$\text{pretok} = \frac{125MHz}{2\text{cikla} * 3\text{barve}} = 20.8Mpik/s \tag{3.4}$$

3.4 Nadaljnje optimizacije in razvoj

Cevovod ima nekaj ozkih grl, ki omejujejo hitrost delovanja. Modula za izračun Haar transformacije in drevesa vsot delujeta samo s hitrostjo 0.5 pike na urin cikel, kar bi bilo potrebno spremeniti na 1 piko na urin cikel. Prav tako sta ta dva algoritma popolnoma neodvisna med bloki, kar kaže na možnost paralelnega procesiranje večih blokov hkrati. Odvisnost med bloki nastane samo pri predikciji in izračunu DC koeficienta.

Ozko grlo predstavlja tudi kodiranje izhodnih podatkov. Sam algoritem v svojem bistvu ne omogoča paralelizacije, saj za izračun kodiranja trenutnega podatka najprej potrebujemo rezultat kodiranja predhodnega podatka.

To ozko grlo izgine, če je uporabljena dovolj izgubna nastavitev kodeka. To pomeni, da je število izhodnih podatkov dovolj majhno, da aritmetično kodiranje ne zavira delovanja predhodnih modulov, kljub temu, da ti delujejo paralelno. Kakšno velikost paralelizacije taka arhitektura dovoljuje, bi bilo v prihodnje potrebno še raziskati.

Tabele za adaptivno kodiranje so v tem delu implementirane kot registri,

kar prinese veliko porabo registrskih rezin in rezin LUT. Učinkovitejša rešitev bi bila uporaba bločnega pomnilnika, ki bi prinesla manjšo zasedenost logike in s tem lažje povezovanje logike po čipu FPGA.

Prav tako bi bilo potrebno popraviti referenčni model, in sicer tako, da bi delovala tudi druga preslikava iz časovnega v frekvenčno območje - DCT s prekrivajočimi bloki. Ta algoritem bi bilo nato potrebno implementirati.

Implementacija ima še veliko možnosti za razvoj - implementirati bi bilo potrebno še drugi tipe slik (slike P in B), ki potrebujejo predikcijo gibanja med posameznimi slikami. Prav tako bi bilo potrebno omogočiti kvantizacijo, ki trenutno ni implementirana zaradi nedelujoče funkcionalnosti referenčnega modela. Naslednji korak bi bil lahko implementacija naprednih funkcij, kot je na primer optimizacija RDO.

Poglavje 4

Sklepne ugotovitve

Trenutno stanje referenčnega modela kodirnika Daala ne omogoča implementacije vseh osnovnih funkcij cevovoda. Kljub temu je implementacija z določenimi omejitvami mogoča.

Algoritmi so primerni za uporabo v čipu FPGA, saj ne vsebujejo elementov, ki jih ni mogoče implementirati z vidika prezapletene logike.

Implementacija deluje zadovoljivo in zasede pričakovano število virov na čipu FPGA. Pretok skozi implementiran cevovod kodirnika je manjši od pričakovanj zaradi potrebe po hkratnem branju in pisanju virov v dveh modulih (Haarova preslikava in drevo vsot).

Referenčni model potrebuje izboljšave predvsem v smislu odprave napak, ki onemogočajo delovanje modela v določenih nastavitvah. Prav tako bi bilo potrebno preučiti delovanje aritmetičnega kodiranja ter se odločiti le za določene vrste vhodnih podatkov - trenutni izbor vhodnih podatkov je namreč prevelik in s tem prezapleten.

Kodirnik Daala je z nekaj spremembami primeren kandidat za uporabo v programabilnih vezjih FPGA in čipih ASIC.

Dodatek A

Izvorna koda implementacije

```
package daala_pkg;

/*Smallest blocks are 4x4*/
parameter OD_LOG_BSIZE0 = 2;
/*There are 5 block sizes total (4x4, 8x8, 16x16, 32x32 and 64
   x64).*/
parameter OD_NBSIZES    = 5;
/*The log of the maximum length of the side of a block.*/
parameter OD_LOG_BSIZE_MAX = OD_LOG_BSIZE0 + OD_NBSIZES - 1;
/*The maximum length of the side of a block.*/
parameter OD_BSIZE_MAX  =  1 << OD_LOG_BSIZE_MAX;
/*The maximum number of quad tree levels when splitting a super
    block.*/
parameter OD_MAX_SB_SPLITS = OD_NBSIZES - 1;

parameter OD_HAAR_BITS = 14;
parameter OD_MAX_TREE_BITS = 16;

const logic[90-1:0][7:0] PVQ_INIT =
{8'd20, 8'd20, 8'd20, 8'd20, 8'd20,
 8'd20, 8'd20, 8'd20, 8'd20, 8'd20,
 8'd20, 8'd20, 8'd20, 8'd20, 8'd20,
 8'd20, 8'd20, 8'd20, 8'd20, 8'd20,
 8'd20, 8'd20, 8'd20, 8'd21, 8'd20,
 8'd20, 8'd20, 8'd22, 8'd20, 8'd26,
 8'd28, 8'd28, 8'd28, 8'd28, 8'd28,
 8'd28, 8'd28, 8'd28, 8'd28, 8'd28,
 8'd28, 8'd28, 8'd28, 8'd28, 8'd28,
 8'd28, 8'd28, 8'd28, 8'd28, 8'd28,
 8'd28, 8'd28, 8'd28, 8'd29, 8'd28,
 8'd28, 8'd28, 8'd31, 8'd28, 8'd36,
```

```
 8'd16, 8'd16, 8'd16, 8'd16, 8'd16,
 8'd16, 8'd16, 8'd16, 8'd16, 8'd16,
 8'd16, 8'd16, 8'd16, 8'd16, 8'd16,
 8'd16, 8'd16, 8'd16, 8'd16, 8'd16,
 8'd16, 8'd16, 8'd16, 8'd17, 8'd16,
 8'd16, 8'd16, 8'd18, 8'd16, 8'd21};

const logic [134:0][15:0] OD_UNIFORM_CDFS_Q15 =
{16'd32768, 16'd30720, 16'd28672, 16'd26624,
 16'd24576, 16'd22528, 16'd20480, 16'd18432,
 16'd16384, 16'd14336, 16'd12288, 16'd10240,
 16'd08192, 16'd06144, 16'd04096, 16'd02048,
 16'd32768, 16'd30583, 16'd28399, 16'd26214,
 16'd24030, 16'd21845, 16'd19661, 16'd17476,
 16'd15292, 16'd13107, 16'd10923, 16'd08738,
 16'd06554, 16'd04369, 16'd02185,
 16'd32768, 16'd30427, 16'd28087, 16'd25746,
 16'd23406, 16'd21065, 16'd18725, 16'd16384,
 16'd14043, 16'd11703, 16'd09362, 16'd07022,
 16'd04681, 16'd02341,
 16'd32768, 16'd30247, 16'd27727, 16'd25206,
 16'd22686, 16'd20165, 16'd17644, 16'd15124,
 16'd12603, 16'd10082, 16'd07562, 16'd05041,
 16'd02521,
 16'd32768, 16'd30037, 16'd27307, 16'd24576,
 16'd21845, 16'd19115, 16'd16384, 16'd13653,
 16'd10923, 16'd08192, 16'd05461, 16'd02731,
 16'd32768, 16'd29789, 16'd26810, 16'd23831,
 16'd20852, 16'd17873, 16'd14895, 16'd11916,
 16'd08937, 16'd05958, 16'd02979,
 16'd32768, 16'd29491, 16'd26214, 16'd22938,
 16'd19661, 16'd16384, 16'd13107, 16'd09830,
 16'd06554, 16'd03277,
 16'd32768, 16'd29127, 16'd25486, 16'd21845,
 16'd18204, 16'd14564, 16'd10923, 16'd07282,
 16'd03641,
 16'd32768, 16'd28672, 16'd24576, 16'd20480,
 16'd16384, 16'd12288, 16'd08192, 16'd04096,
 16'd32768, 16'd28087, 16'd23406, 16'd18725,
 16'd14043, 16'd09362, 16'd04681,
 16'd32768, 16'd27307, 16'd21845, 16'd16384,
 16'd10923, 16'd05461,
 16'd32768, 16'd26214, 16'd19661, 16'd13107,
 16'd06554,
 16'd32768, 16'd24576, 16'd16384, 16'd08192,
 16'd32768, 16'd21845, 16'd10923,
 16'd32768, 16'd16384};
```

```
parameter OD_EC_UINT_BITS = 4;
parameter OD_EC_WINDOW_SIZE = 32;

parameter OD_EC_STORAGE = 1024*64;

parameter OD_ADAPT_HAAR_INCREMENT = 128;
parameter OD_ADAPT_DC_INCREMENT = 128;

function [15:0] daala_min_16 (input logic [15:0] a, input logic
    [15:0] b);
begin
  daala_min_16 = (a < b) ? a : b;
end
endfunction

function [5:0] daala_min_6 (input logic [5:0] a, input logic
   [5:0] b);
begin
  daala_min_6 = (a < b) ? a : b;
end
endfunction

function [4:0] daala_min_5 (input logic [4:0] a, input logic
   [4:0] b);
begin
  daala_min_5 = (a < b) ? a : b;
end
endfunction

function [4:0] daala_max_5 (input logic [4:0] a, input logic
   [4:0] b);
begin
  daala_max_5 = (a > b) ? a : b;
end
endfunction

function [15:0] daala_max_16 (input logic [15:0] a, input logic
    [15:0] b);
begin
  daala_max_16 = (a > b) ? a : b;
end
endfunction

function [15:0] daala_max_16s (input logic [15:0] a, input
   logic [15:0] b);
begin
```

```
  daala_max_16s = ($signed(a) > $signed(b)) ? a : b;
end
endfunction

endpackage

import daala_pkg::*;

module daala_top (
  input  logic        clk,
  input  logic        rst,

  input  logic [10:0] w,
  input  logic [10:0] h,

  input  logic        in_vld,
  input  logic [ 7:0] in_dat_y,
  input  logic [ 7:0] in_dat_u,
  input  logic [ 7:0] in_dat_v,
  input  logic        in_lst,
  output logic        in_rdy,

  output logic        out_vld,
  output logic [31:0] out_dat,
  output logic [ 3:0] out_keep,
  output logic        out_lst,
  input  logic        out_rdy
);

logic [31:0] entenc_low;
logic [15:0] entenc_rng;
logic [15:0] entenc_offs;
logic [ 4:0] entenc_cnt;
logic [ 5:0] entenc_nend_bits;
logic [31:0] entenc_end_window;

logic        sof;

logic        reorder_y_vld;
logic        reorder_y_lst;
logic [ 7:0] reorder_y_dat;
logic        reorder_y_rdy;

logic        reorder_u_vld;
logic        reorder_u_lst;
logic [ 7:0] reorder_u_dat;
logic        reorder_u_rdy;
```

```
logic         reorder_v_vld;
logic         reorder_v_lst;
logic [ 7:0] reorder_v_dat;
logic         reorder_v_rdy;

logic         haar_switch_vld;
logic         haar_switch_lst;
logic [ 7:0] haar_switch_dat;
logic [ 2:0] haar_switch_ln ;
logic [ 1:0] haar_switch_pli;
logic [ 4:0] haar_switch_sby;
logic [ 4:0] haar_switch_sbx;
logic         haar_switch_rdy;

logic         haar_vld;
logic [ 2:0] haar_ln ;
logic [ 1:0] haar_pli;
logic [ 4:0] haar_sby;
logic [OD_HAAR_BITS-1:0] haar_dc;
logic [ 4:0] haar_sbx;

logic         sb_buff_haar_en  ;
logic         sb_buff_haar_wea ;
logic         sb_buff_haar_web ;
logic [11:0] sb_buff_haar_adra;
logic [11:0] sb_buff_haar_adrb;
logic [15:0] sb_buff_haar_dina;
logic [15:0] sb_buff_haar_dinb;

logic         sb_buff_max_tree_en  ;
logic [11:0] sb_buff_max_tree_adr ;
logic [15:0] sb_buff_max_tree_dout;

logic         sb_buff_max_tree_entenc_en;
logic [11:0] sb_buff_max_tree_entenc_adr ;
logic [15:0] sb_buff_max_tree_entenc_dout;

logic         sb_buff_switch;

logic                      haar_quant_vld;
logic [OD_HAAR_BITS-1:0] haar_quant_quant;
logic [              1:0] haar_quant_pli;
logic                      haar_quant_rdy;

logic         haar_dc_vld;
logic [ 2:0] haar_dc_ln ;
```

```
logic [ 1:0] haar_dc_pli;
logic [ 4:0] haar_dc_sby;
logic [ 4:0] haar_dc_sbx;
logic        haar_dc_rdy;

logic        gen_enc_cdf_adapt_vld;
logic [15:0] gen_enc_cdf_adapt_fl ;
logic [15:0] gen_enc_cdf_adapt_fh ;
logic [15:0] gen_enc_cdf_adapt_ft ;
logic        gen_enc_cdf_adapt_rdy;

logic        gen_enc_vld;
logic [15:0] gen_enc_x  ;
logic [15:0] gen_enc_decay;
logic        gen_enc_rdy;

logic        gen_enc_bits_vld;
logic [31:0] gen_enc_bits_fl ;
logic [ 4:0] gen_enc_bits_ftb;
logic        gen_enc_bits_rdy;


logic        laplace_enc_cdf_adapt_vld;
logic [15:0] laplace_enc_cdf_adapt_fl ;
logic [15:0] laplace_enc_cdf_adapt_fh ;
logic [15:0] laplace_enc_cdf_adapt_ft ;
logic        laplace_enc_cdf_adapt_rdy;

logic        laplace_enc_q15_vld;
logic [15:0] laplace_enc_q15_fl ;
logic [15:0] laplace_enc_q15_fh ;
logic        laplace_enc_q15_rdy;

logic        laplace_enc_bits_vld;
logic [31:0] laplace_enc_bits_fl ;
logic [ 4:0] laplace_enc_bits_ftb;
logic        laplace_enc_bits_rdy;


logic        sb_tree_sum_buff_en  ;
logic [ 1:0] sb_tree_sum_buff_wea ;
logic [10:0] sb_tree_sum_buff_adra;
logic [10:0] sb_tree_sum_buff_adrb;
logic [31:0] sb_tree_sum_buff_dina;
logic [31:0] sb_tree_sum_buff_douta;
logic [31:0] sb_tree_sum_buff_doutb;
```

```
logic         max_tree_vld;
logic [ 2:0] max_tree_ln ;
logic [ 1:0] max_tree_pli;
logic         max_tree_rdy;


logic       eof;

logic         sb_tree_sum_buff_entenc_en  ;
logic [10:0] sb_tree_sum_buff_entenc_adra;
logic [10:0] sb_tree_sum_buff_entenc_adrb;
logic [31:0] sb_tree_sum_buff_entenc_douta;
logic [31:0] sb_tree_sum_buff_entenc_doutb;


logic       max_tree_entenc_vld;
logic [3:0] max_tree_entenc_val;
logic [1:0] max_tree_entenc_cdf_sel;
logic [8:0] max_tree_entenc_cdf_num;
logic [4:0] max_tree_entenc_nsyms;
logic       max_tree_entenc_rdy;

logic         max_tree_entenc_bits_vld;
logic [31:0] max_tree_entenc_bits_fl ;
logic [ 4:0] max_tree_entenc_bits_ftb;
logic         max_tree_entenc_bits_rdy;


reorder reorder_i (
  .clk         (clk),
  .rst         (rst),

  .w           (w),
  .h           (h),

  .sof         (sof),

  .in_vld      (in_vld  ),
  .in_dat_y    (in_dat_y),
  .in_dat_u    (in_dat_u),
  .in_dat_v    (in_dat_v),
  .in_lst      (in_lst  ),
  .in_rdy      (in_rdy  ),

  .out_y_vld   (reorder_y_vld),
  .out_y_lst   (reorder_y_lst),
  .out_y_dat   (reorder_y_dat),
```

```
  .out_y_rdy  (reorder_y_rdy),

  .out_u_vld  (reorder_u_vld),
  .out_u_lst  (reorder_u_lst),
  .out_u_dat  (reorder_u_dat),
  .out_u_rdy  (reorder_u_rdy),

  .out_v_vld  (reorder_v_vld),
  .out_v_lst  (reorder_v_lst),
  .out_v_dat  (reorder_v_dat),
  .out_v_rdy  (reorder_v_rdy)
);


haar_switch haar_switch_i (
  .clk        (clk),
  .rst        (rst),

  .sof        (sof),
  .nhsb       (w[10:6]),
  .nvsb       (h[10:6]),

  .in_y_vld  (reorder_y_vld),
  .in_y_dat  (reorder_y_dat),
  .in_y_lst  (reorder_y_lst),
  .in_y_rdy  (reorder_y_rdy),

  .in_u_vld  (reorder_u_vld),
  .in_u_dat  (reorder_u_dat),
  .in_u_lst  (reorder_u_lst),
  .in_u_rdy  (reorder_u_rdy),

  .in_v_vld  (reorder_v_vld),
  .in_v_dat  (reorder_v_dat),
  .in_v_lst  (reorder_v_lst),
  .in_v_rdy  (reorder_v_rdy),

  .out_vld   (haar_switch_vld),
  .out_dat   (haar_switch_dat),
  .out_lst   (haar_switch_lst),
  .out_ln    (haar_switch_ln ),
  .out_pli   (haar_switch_pli),
  .out_sby   (haar_switch_sby),
  .out_sbx   (haar_switch_sbx),
  .out_rdy   (haar_switch_rdy)
);
```

```
sb_buff sb_buff_i (
  .clk                  (clk),
  .rst                  (rst),

  .sof                  (sof),
  .nhsb                 (w[10:6]),
  .nvsb                 (h[10:6]),

  .haar_done            (haar_vld),
  .haar_en              (sb_buff_haar_en  ),
  .haar_wea             (sb_buff_haar_wea ),
  .haar_web             (sb_buff_haar_web ),
  .haar_adra            (sb_buff_haar_adra),
  .haar_adrb            (sb_buff_haar_adrb),
  .haar_dina            (sb_buff_haar_dina),
  .haar_dinb            (sb_buff_haar_dinb),

  .max_tree_done        (max_tree_vld),
  .max_tree_en          (sb_buff_max_tree_en  ),
  .max_tree_adr         (sb_buff_max_tree_adr ),
  .max_tree_dout        (sb_buff_max_tree_dout),

  .max_tree_entenc_done (max_tree_rdy),
  .max_tree_entenc_en   (sb_buff_max_tree_entenc_en  ),
  .max_tree_entenc_adr  (sb_buff_max_tree_entenc_adr ),
  .max_tree_entenc_dout (sb_buff_max_tree_entenc_dout),

  .sb_buff_switch       (sb_buff_switch)
);



haar haar_i (
  .clk      (clk),
  .rst      (rst),

  .in_vld   (haar_switch_vld),
  .in_lst   (haar_switch_lst),
  .in_ln    (haar_switch_ln ),
  .in_pli   (haar_switch_pli),
  .in_sbx   (haar_switch_sby),
  .in_sby   (haar_switch_sbx),
  .in_x     (haar_switch_dat),
  .in_rdy   (haar_switch_rdy),

  .sb_buff_en   (sb_buff_haar_en  ),
```

```
  .sb_buff_wea  (sb_buff_haar_wea ),
  .sb_buff_web  (sb_buff_haar_web ),
  .sb_buff_adra (sb_buff_haar_adra),
  .sb_buff_adrb (sb_buff_haar_adrb),
  .sb_buff_dina (sb_buff_haar_dina),
  .sb_buff_dinb (sb_buff_haar_dinb),

  .out_vld  (haar_vld),
  .out_dc   (haar_dc ),
  .out_ln   (haar_ln ),
  .out_pli  (haar_pli),
  .out_sbx  (haar_sby),
  .out_sby  (haar_sbx),
  .out_rdy  (sb_buff_switch)
);

// dc coeff encode
haar_dc_sb haar_dc_sb_i (
  .clk,
  .rst,

  .nhsb       (w[10:6]),

  .in_vld     (haar_vld && sb_buff_switch),
  .in_dc      (haar_dc ),
  .in_ln      (haar_ln ),
  .in_pli     (haar_pli),
  .in_sbx     (haar_sby),
  .in_sby     (haar_sbx),

  .quant_vld  (haar_quant_vld  ),
  .quant_dc   (haar_quant_quant),
  .quant_pli  (haar_quant_pli  ),
  .quant_rdy  (haar_quant_rdy  ),

  .out_vld    (haar_dc_vld),
  .out_ln     (haar_dc_ln ),
  .out_pli    (haar_dc_pli),
  .out_sbx    (haar_dc_sby),
  .out_sby    (haar_dc_sbx),
  .out_rdy    (haar_dc_rdy)
);

gen_enc gen_enc_i (
  .rst            (rst),
  .clk            (clk),
```

```
  .sof            (sof),

  .in_vld         (haar_quant_vld  ),
  .in_x           (haar_quant_quant),
  .in_pli         (haar_quant_pli  ),
  .in_rdy         (haar_quant_rdy  ),

  .out_vld        (gen_enc_cdf_adapt_vld),
  .out_fl         (gen_enc_cdf_adapt_fl ),
  .out_fh         (gen_enc_cdf_adapt_fh ),
  .out_ft         (gen_enc_cdf_adapt_ft ),
  .out_rdy        (gen_enc_cdf_adapt_rdy),

  .laplace_vld    (gen_enc_vld  ),
  .laplace_x      (gen_enc_x    ),
  .laplace_decay  (gen_enc_decay),
  .laplace_rdy    (gen_enc_rdy  ),

  .bits_vld       (gen_enc_bits_vld),
  .bits_ft        (gen_enc_bits_fl ),
  .bits_ftb       (gen_enc_bits_ftb),
  .bits_rdy       (gen_enc_bits_rdy)
);

laplace_enc laplace_enc_i (
  .rst        (rst),
  .clk        (clk),

  .in_vld     (gen_enc_vld  ),
  .in_x       (gen_enc_x    ),
  .in_decay   (gen_enc_decay),
  .in_rdy     (gen_enc_rdy  ),

  .out_cdf_vld (laplace_enc_cdf_adapt_vld),
  .out_cdf_fl  (laplace_enc_cdf_adapt_fl ),
  .out_cdf_fh  (laplace_enc_cdf_adapt_fh ),
  .out_cdf_ft  (laplace_enc_cdf_adapt_ft ),
  .out_cdf_rdy (laplace_enc_cdf_adapt_rdy),

  .out_q15_vld (laplace_enc_q15_vld),
  .out_q15_fl  (laplace_enc_q15_fl ),
  .out_q15_fh  (laplace_enc_q15_fh ),
  .out_q15_rdy (laplace_enc_q15_rdy),

  .bits_vld   (laplace_enc_bits_vld),
  .bits_ft    (laplace_enc_bits_fl ),
  .bits_ftb   (laplace_enc_bits_ftb),
```

```
  .bits_rdy   (laplace_enc_bits_rdy)
);


// max tree
max_tree max_tree_i (
  .clk          (clk),
  .rst          (rst),

  .in_vld       (haar_dc_vld),
  .in_ln        (haar_dc_ln ),
  .in_pli       (haar_dc_pli),
  .in_rdy       (haar_dc_rdy),

  .sb_coef_buff_en        (sb_buff_max_tree_en  ),
  .sb_coef_buff_adr       (sb_buff_max_tree_adr ),
  .sb_coef_buff_dout      (sb_buff_max_tree_dout),

  .sb_tree_sum_buff_en    (sb_tree_sum_buff_en   ),
  .sb_tree_sum_buff_wea   (sb_tree_sum_buff_wea  ),
  .sb_tree_sum_buff_adra  (sb_tree_sum_buff_adra ),
  .sb_tree_sum_buff_adrb  (sb_tree_sum_buff_adrb ),
  .sb_tree_sum_buff_dina  (sb_tree_sum_buff_dina ),
  .sb_tree_sum_buff_douta (sb_tree_sum_buff_douta),
  .sb_tree_sum_buff_doutb (sb_tree_sum_buff_doutb),

  .out_vld      (max_tree_vld),
  .out_ln       (max_tree_ln ),
  .out_pli      (max_tree_pli),
  .out_rdy      (sb_buff_switch)
);

max_tree_entenc max_tree_entenc_i (
  .clk          (clk),
  .rst          (rst),

  .sof          (sof),
  .nhsb         (w[10:6]),
  .nvsb         (h[10:6]),
  .eof          (eof),

  .in_vld       (max_tree_vld && sb_buff_switch),
  .in_ln        (max_tree_ln ),
  .in_pli       (max_tree_pli),
  .in_rdy       (max_tree_rdy),

  .sb_coef_buff_en        (sb_buff_max_tree_entenc_en  ),
```

```
  .sb_coef_buff_adr       (sb_buff_max_tree_entenc_adr ),
  .sb_coef_buff_dout      (sb_buff_max_tree_entenc_dout),

  .sb_tree_sum_buff_en    (sb_tree_sum_buff_entenc_en   ),
  .sb_tree_sum_buff_adra  (sb_tree_sum_buff_entenc_adra ),
  .sb_tree_sum_buff_adrb  (sb_tree_sum_buff_entenc_adrb ),
  .sb_tree_sum_buff_douta (sb_tree_sum_buff_entenc_douta),
  .sb_tree_sum_buff_doutb (sb_tree_sum_buff_entenc_doutb),

  .out_vld      (max_tree_entenc_vld    ),
  .out_val      (max_tree_entenc_val    ),
  .out_cdf_sel  (max_tree_entenc_cdf_sel),
  .out_cdf_num  (max_tree_entenc_cdf_num),
  .out_nsyms    (max_tree_entenc_nsyms  ),
  .out_rdy      (max_tree_entenc_rdy    ),

  .bits_vld (max_tree_entenc_bits_vld),
  .bits_fl  (max_tree_entenc_bits_fl ),
  .bits_ftb (max_tree_entenc_bits_ftb),
  .bits_rdy (max_tree_entenc_bits_rdy)
);

sb_tree_sum_buff sb_tree_sum_buff_i (
  .clk                  (clk),
  .rst                  (rst),

  .max_tree_done        (max_tree_vld),
  .max_tree_en          (sb_tree_sum_buff_en  ),
  .max_tree_wea         (sb_tree_sum_buff_wea ),
  .max_tree_adra        (sb_tree_sum_buff_adra),
  .max_tree_adrb        (sb_tree_sum_buff_adrb),
  .max_tree_dina        (sb_tree_sum_buff_dina),
  .max_tree_douta       (sb_tree_sum_buff_douta),
  .max_tree_doutb       (sb_tree_sum_buff_doutb),

  .max_tree_entenc_done  (max_tree_rdy),
  .max_tree_entenc_en    (sb_tree_sum_buff_entenc_en  ),
  .max_tree_entenc_adra  (sb_tree_sum_buff_entenc_adra ),
  .max_tree_entenc_adrb  (sb_tree_sum_buff_entenc_adrb ),
  .max_tree_entenc_douta (sb_tree_sum_buff_entenc_douta),
  .max_tree_entenc_doutb (sb_tree_sum_buff_entenc_doutb)
);



// Entropy encoder
logic        entenc_ctrl_bool_q15_vld;
```

```
logic         entenc_ctrl_bool_q15_val;
logic [15:0] entenc_ctrl_bool_q15_fz ;
logic         entenc_ctrl_bool_q15_rdy;

logic         entenc_ctrl_uint_vld;
logic [31:0] entenc_ctrl_uint_fl ;
logic [31:0] entenc_ctrl_uint_ft ;
logic         entenc_ctrl_uint_rdy;

logic         entenc_ctrl_bits_vld;
logic [31:0] entenc_ctrl_bits_fl ;
logic [ 4:0] entenc_ctrl_bits_ftb;
logic         entenc_ctrl_bits_rdy;

entenc_ctrl entenc_ctrl_i (
  .rst           (rst),
  .clk           (clk),

  .sof           (sof),

  .bool_q15_vld (entenc_ctrl_bool_q15_vld),
  .bool_q15_val (entenc_ctrl_bool_q15_val),
  .bool_q15_fz  (entenc_ctrl_bool_q15_fz ),
  .bool_q15_rdy (entenc_ctrl_bool_q15_rdy),

  .uint_vld     (entenc_ctrl_uint_vld),
  .uint_fl      (entenc_ctrl_uint_fl ),
  .uint_ft      (entenc_ctrl_uint_ft ),
  .uint_rdy     (entenc_ctrl_uint_rdy),

  .bits_vld     (entenc_ctrl_bits_vld),
  .bits_fl      (entenc_ctrl_bits_fl ),
  .bits_ftb     (entenc_ctrl_bits_ftb),
  .bits_rdy     (entenc_ctrl_bits_rdy)
);

logic         entenc_uint_vld;
logic [15:0] entenc_uint_fl;
logic [15:0] entenc_uint_fh;
logic         entenc_uint_rdy;

logic         entenc_uint_bits_vld;
logic [31:0] entenc_uint_bits_fl ;
logic [ 4:0] entenc_uint_bits_ftb;
logic         entenc_uint_bits_rdy;

entenc_uint entenc_uint_i (
```

```
  .rst      (rst),
  .clk      (clk),

  .in_vld   (entenc_ctrl_uint_vld),
  .in_fl    (entenc_ctrl_uint_fl ),
  .in_ft    (entenc_ctrl_uint_ft ),
  .in_rdy   (entenc_ctrl_uint_rdy),

  .out_vld  (entenc_uint_vld),
  .out_fl   (entenc_uint_fl),
  .out_fh   (entenc_uint_fh),
  .out_rdy  (entenc_uint_rdy),

  .bits_vld (entenc_uint_bits_vld),
  .bits_fl  (entenc_uint_bits_fl ),
  .bits_ftb (entenc_uint_bits_ftb),
  .bits_rdy (entenc_uint_bits_rdy)
);


logic         max_tree_cdf_adapt_vld;
logic [15:0] max_tree_cdf_adapt_fl ;
logic [15:0] max_tree_cdf_adapt_fh ;
logic [15:0] max_tree_cdf_adapt_ft ;
logic         max_tree_cdf_adapt_rdy;

entenc_cdf_adapt entenc_cdf_adapt_i (
  .rst          (rst),
  .clk          (clk),

  .sof          (sof),

  .in_vld       (max_tree_entenc_vld    ),
  .in_val       (max_tree_entenc_val    ),
  .in_cdf_sel   (max_tree_entenc_cdf_sel),
  .in_cdf_num   (max_tree_entenc_cdf_num),
  .in_nsyms     (max_tree_entenc_nsyms  ),
  .in_rdy       (max_tree_entenc_rdy    ),

  .out_vld      (max_tree_cdf_adapt_vld),
  .out_fl       (max_tree_cdf_adapt_fl ),
  .out_fh       (max_tree_cdf_adapt_fh ),
  .out_ft       (max_tree_cdf_adapt_ft ),
  .out_rdy      (max_tree_cdf_adapt_rdy)
);

logic         entenc_cdf_adapt_vld;
```

```
logic [15:0] entenc_cdf_adapt_fl ;
logic [15:0] entenc_cdf_adapt_fh ;
logic [15:0] entenc_cdf_adapt_ft ;
logic        entenc_cdf_adapt_rdy;

entenc_cdf_adapt_switch entenc_cdf_adapt_switch_i (
  .rst                (rst),
  .clk                (clk),

  .in_gen_enc_vld     (gen_enc_cdf_adapt_vld),
  .in_gen_enc_fl      (gen_enc_cdf_adapt_fl ),
  .in_gen_enc_fh      (gen_enc_cdf_adapt_fh ),
  .in_gen_enc_ft      (gen_enc_cdf_adapt_ft ),
  .in_gen_enc_rdy     (gen_enc_cdf_adapt_rdy),

  .in_laplace_enc_vld (laplace_enc_cdf_adapt_vld),
  .in_laplace_enc_fl  (laplace_enc_cdf_adapt_fl ),
  .in_laplace_enc_fh  (laplace_enc_cdf_adapt_fh ),
  .in_laplace_enc_ft  (laplace_enc_cdf_adapt_ft ),
  .in_laplace_enc_rdy (laplace_enc_cdf_adapt_rdy),

  .in_max_tree_vld    (max_tree_cdf_adapt_vld),
  .in_max_tree_fl     (max_tree_cdf_adapt_fl ),
  .in_max_tree_fh     (max_tree_cdf_adapt_fh ),
  .in_max_tree_ft     (max_tree_cdf_adapt_ft ),
  .in_max_tree_rdy    (max_tree_cdf_adapt_rdy),

  .out_vld            (entenc_cdf_adapt_vld),
  .out_fl             (entenc_cdf_adapt_fl ),
  .out_fh             (entenc_cdf_adapt_fh ),
  .out_ft             (entenc_cdf_adapt_ft ),
  .out_rdy            (entenc_cdf_adapt_rdy)
);



logic        entenc_calc_vld;
logic [31:0] entenc_calc_low;
logic [15:0] entenc_calc_rng;
logic        entenc_calc_rdy;

logic        entenc_calc_q15_rdy;

assign  entenc_uint_rdy = entenc_uint_vld &&
   entenc_calc_q15_rdy;
assign  laplace_enc_q15_rdy = laplace_enc_q15_vld &&
   entenc_calc_q15_rdy;
```

```
entenc_calc entenc_calc_i (
  .rst              (rst),
  .clk              (clk),

  .entenc_low       (entenc_low),
  .entenc_rng       (entenc_rng),

  .in_bool_q15_vld  (entenc_ctrl_bool_q15_vld),
  .in_bool_q15_val  (entenc_ctrl_bool_q15_val),
  .in_bool_q15_fz   (entenc_ctrl_bool_q15_fz ),
  .in_bool_q15_rdy  (entenc_ctrl_bool_q15_rdy),

  .in_q15_vld       (entenc_uint_vld || laplace_enc_q15_vld),
  .in_q15_fl        (entenc_uint_vld ? entenc_uint_fl :
     laplace_enc_q15_fl ),
  .in_q15_fh        (entenc_uint_vld ? entenc_uint_fh :
     laplace_enc_q15_fh ),
  .in_q15_rdy       (entenc_calc_q15_rdy),

  .in_cdf_adapt_vld (entenc_cdf_adapt_vld),
  .in_cdf_adapt_fl  (entenc_cdf_adapt_fl ),
  .in_cdf_adapt_fh  (entenc_cdf_adapt_fh ),
  .in_cdf_adapt_ft  (entenc_cdf_adapt_ft ),
  .in_cdf_adapt_rdy (entenc_cdf_adapt_rdy),

  .out_vld          (entenc_calc_vld),
  .out_low          (entenc_calc_low),
  .out_rng          (entenc_calc_rng),
  .out_rdy          (entenc_calc_rdy)
);

logic         entenc_norm_en;
logic         entenc_norm_bs;
logic [15:0] entenc_norm_adr;
logic [31:0] entenc_norm_dat_w;

entenc_norm entenc_norm_i (
  .rst      (rst),
  .clk      (clk),

  .in_vld (entenc_calc_vld),
  .in_low (entenc_calc_low),
  .in_rng (entenc_calc_rng),
  .in_rdy (entenc_calc_rdy),
```

```
  .pc_buff_en    (entenc_norm_en   ),
  .pc_buff_adr   (entenc_norm_adr  ),
  .pc_buff_bs    (entenc_norm_bs   ),
  .pc_buff_dat_w (entenc_norm_dat_w),

  .low      (entenc_low),
  .rng      (entenc_rng),
  .cnt      (entenc_cnt),
  .offs     (entenc_offs)
);

logic         entenc_bits_switch_vld;
logic [31:0] entenc_bits_switch_fl ;
logic [ 4:0] entenc_bits_switch_ftb;
logic         entenc_bits_switch_rdy;

entenc_bits_switch entenc_bits_switch_i (
  .rst              (rst),
  .clk              (clk),

  .in_ctrl_vld     (entenc_ctrl_bits_vld),
  .in_ctrl_fl      (entenc_ctrl_bits_fl ),
  .in_ctrl_ftb     (entenc_ctrl_bits_ftb),
  .in_ctrl_rdy     (entenc_ctrl_bits_rdy),

  .in_max_tree_vld (max_tree_entenc_bits_vld),
  .in_max_tree_fl  (max_tree_entenc_bits_fl ),
  .in_max_tree_ftb (max_tree_entenc_bits_ftb),
  .in_max_tree_rdy (max_tree_entenc_bits_rdy),

  .in_uint_vld     (entenc_uint_bits_vld),
  .in_uint_fl      (entenc_uint_bits_fl ),
  .in_uint_ftb     (entenc_uint_bits_ftb),
  .in_uint_rdy     (entenc_uint_bits_rdy),

  .in_gen_enc_vld  (gen_enc_bits_vld),
  .in_gen_enc_fl   (gen_enc_bits_fl ),
  .in_gen_enc_ftb  (gen_enc_bits_ftb),
  .in_gen_enc_rdy  (gen_enc_bits_rdy),

  .in_laplace_enc_vld  (laplace_enc_bits_vld),
  .in_laplace_enc_fl   (laplace_enc_bits_fl ),
  .in_laplace_enc_ftb  (laplace_enc_bits_ftb),
  .in_laplace_enc_rdy  (laplace_enc_bits_rdy),

  .out_vld         (entenc_bits_switch_vld),
  .out_fl          (entenc_bits_switch_fl ),
```

```
  .out_ftb         (entenc_bits_switch_ftb),
  .out_rdy         (entenc_bits_switch_rdy)
);


logic         entenc_bits_vld ;
logic [31:0] entenc_bits_dat ;
logic [ 3:0] entenc_bits_keep;
logic         entenc_bits_rdy ;

entenc_bits entenc_bits_i (
  .rst      (rst),
  .clk      (clk),

  .nend_bits  (entenc_nend_bits),
  .end_window (entenc_end_window),

  .in_vld   (entenc_bits_switch_vld),
  .in_fl    (entenc_bits_switch_fl ),
  .in_ftb   (entenc_bits_switch_ftb),
  .in_rdy   (entenc_bits_switch_rdy),

  .out_vld  (entenc_bits_vld ),
  .out_dat  (entenc_bits_dat ),
  .out_keep (entenc_bits_keep),
  .out_rdy  (entenc_bits_rdy )
);

logic         entenc_done_en;
logic         entenc_done_wr;
logic [15:0] entenc_done_adr;
logic [15:0] entenc_done_dat_r;
logic [15:0] entenc_done_dat_w;

entenc_buff entenc_buff_i (
  .clk        (clk),

  .norm_en    (entenc_norm_en   ),
  .norm_bs    (entenc_norm_bs   ),
  .norm_adr   (entenc_norm_adr  ),
  .norm_dat_w (entenc_norm_dat_w),

  .done_en    (entenc_done_en   ),
  .done_we    (entenc_done_we   ),
  .done_adr   (entenc_done_adr  ),
  .done_dat_r (entenc_done_dat_r),
  .done_dat_w (entenc_done_dat_w)
```

```
);

entenc_done entenc_done_i (
  .rst      (rst),
  .clk      (clk),

  .in_vld  (entenc_bits_vld ),
  .in_dat  (entenc_bits_dat ),
  .in_keep (entenc_bits_keep),
  .in_rdy  (entenc_bits_rdy ),

  .eof_vld        (eof),
  .eof_low        (entenc_low),
  .eof_rng        (entenc_rng),
  .eof_cnt        (entenc_cnt),
  .eof_offs       (entenc_offs),
  .eof_nend_bits  (entenc_nend_bits),
  .eof_end_window (entenc_end_window),

  .pc_buff_en    (entenc_done_en   ),
  .pc_buff_we    (entenc_done_we   ),
  .pc_buff_adr   (entenc_done_adr  ),
  .pc_buff_dat_r (entenc_done_dat_r),
  .pc_buff_dat_w (entenc_done_dat_w),

  .out_vld  (out_vld ),
  .out_dat  (out_dat ),
  .out_keep (out_keep),
  .out_lst  (out_lst ),
  .out_rdy  (out_rdy )
);

endmodule

import daala_pkg::*;

module reorder (
  input  logic        clk,
  input  logic        rst,

  input  logic [10:0] w,
  input  logic [10:0] h,

  output logic        sof,

  input  logic        in_vld,
  input  logic        in_lst,
```

```
  input  logic [ 7:0] in_dat_y,
  input  logic [ 7:0] in_dat_u,
  input  logic [ 7:0] in_dat_v,
  output logic        in_rdy,

  output logic        out_y_vld,
  output logic [ 7:0] out_y_dat,
  output logic        out_y_lst,
  input  logic        out_y_rdy,

  output logic        out_u_vld,
  output logic [ 7:0] out_u_dat,
  output logic        out_u_lst,
  input  logic        out_u_rdy,

  output logic        out_v_vld,
  output logic [ 7:0] out_v_dat,
  output logic        out_v_lst,
  input  logic        out_v_rdy
);

logic        in_trn;
logic [10:0] in_cnt_x;
logic [10:0] in_cnt_y;

logic        out_y_trn;
logic [10:0] out_y_cnt_x;
logic [10:0] out_y_cnt_y;
logic        out_u_trn;
logic [10:0] out_u_cnt_x;
logic [10:0] out_u_cnt_y;
logic        out_v_trn;
logic [10:0] out_v_cnt_x;
logic [10:0] out_v_cnt_y;

assign in_trn = in_vld && in_rdy;

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    in_cnt_x <= #1 'd0;
    in_cnt_y <= #1 'd0;
  end else if (in_trn) begin
    in_cnt_x <= #1 in_cnt_x + 'd1;

    if (in_cnt_x == w - 'd1) begin
      in_cnt_x <= #1 'd0;
      in_cnt_y <= #1 in_cnt_y + 'd1;
```

```
      if (in_cnt_y == h - 'd1) begin
        in_cnt_y <= #1 'd0;
      end
    end
  end
end

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    in_rdy <= #1 'd1;
  else if (in_trn && (in_cnt_y[5:0] == (OD_BSIZE_MAX - 'd1)) &&
      (in_cnt_x == w - 'd1))
    in_rdy <= #1 'd0;
  else if (out_v_trn && out_v_lst && (out_v_cnt_x == w - 'd2))
    in_rdy <= #1 'd1;
end

assign sof = in_trn && (in_cnt_x == 'd0) && (in_cnt_y == 'd0);

reorder_buff reorder_buff_y_i (
 .clka  (clk),
 .ena   (in_trn),
 .wea   (1'b1),
 .addra ({in_cnt_y[5:0], in_cnt_x}),
 .dina  (in_dat_y),

 .clkb  (clk),
 .addrb ({out_y_cnt_y[5:0], out_y_cnt_x}),
 .doutb (out_y_dat)
);

reorder_buff reorder_buff_u_i (
 .clka  (clk),
 .ena   (in_trn && !in_cnt_x[0]),
 .wea   (1'b1),
 .addra ({in_cnt_y[5:0], in_cnt_x}),
 .dina  (in_dat_u),

 .clkb  (clk),
 .addrb ({out_u_cnt_y[5:0], out_u_cnt_x}),
 .doutb (out_u_dat)
);

reorder_buff reorder_buff_v_i (
 .clka  (clk),
 .ena   (in_trn && !in_cnt_x[0]),
```

```
 .wea   (1'b1),
 .addra ({in_cnt_y[5:0], in_cnt_x}),
 .dina  (in_dat_v),

 .clkb  (clk),
 .addrb ({out_v_cnt_y[5:0], out_v_cnt_x}),
 .doutb (out_v_dat)
);


// out_y
assign out_y_trn = ((in_cnt_y > out_y_cnt_y) || (!in_rdy && |
   out_y_cnt_y)) && out_y_rdy;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_y_vld <= #1 'd0;
  else if (out_y_trn)
    out_y_vld <= #1 'd1;
  else if (out_y_rdy)
    out_y_vld <= #1 'd0;
end

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    out_y_cnt_x <= #1 'd0;
    out_y_cnt_y <= #1 'd0;
  end else if (out_y_trn) begin
    out_y_cnt_x <= #1 out_y_cnt_x + 'd1;

    if (out_y_cnt_x[5:0] == (OD_BSIZE_MAX - 1)) begin
      out_y_cnt_x <= #1 out_y_cnt_x - (OD_BSIZE_MAX - 1);
      out_y_cnt_y <= #1 out_y_cnt_y + 'd1;

      if (out_y_cnt_y[5:0] == (OD_BSIZE_MAX - 1)) begin
        if (out_y_cnt_x == w - 'd1) begin
          out_y_cnt_y <= #1 out_y_cnt_y + 'd1;
          out_y_cnt_x <= #1 'd0;
        end else begin
          out_y_cnt_y <= #1 out_y_cnt_y - (OD_BSIZE_MAX - 1);
          out_y_cnt_x <= #1 out_y_cnt_x + 'd1;
        end
      end
    end
  end
end
```

```
always_ff @(posedge clk) begin
  if (out_y_trn)
    out_y_lst <= #1 (out_y_cnt_y[5:0] == (OD_BSIZE_MAX - 1)) &&
        (out_y_cnt_x[5:0] == (OD_BSIZE_MAX - 1));
end

// out_u
assign out_u_trn = ((in_cnt_y > out_u_cnt_y) || (!in_rdy && |
   out_u_cnt_y)) && out_u_rdy;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_u_vld <= #1 'd0;
  else if (out_u_trn)
    out_u_vld <= #1 'd1;
  else if (out_u_rdy)
    out_u_vld <= #1 'd0;
end

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    out_u_cnt_x <= #1 'd0;
    out_u_cnt_y <= #1 'd0;
  end else if (out_u_trn) begin
    out_u_cnt_x <= #1 out_u_cnt_x + 'd2;

    if (out_u_cnt_x[5:0] == (OD_BSIZE_MAX - 2)) begin
      out_u_cnt_x <= #1 out_u_cnt_x - (OD_BSIZE_MAX - 2);
      out_u_cnt_y <= #1 out_u_cnt_y + 'd2;

      if (out_u_cnt_y[5:0] == (OD_BSIZE_MAX/2 - 1)) begin
        if (out_u_cnt_x == w - 'd2) begin
          out_u_cnt_y <= #1 out_u_cnt_y + 'd2;
          out_u_cnt_x <= #1 'd0;
        end else begin
          out_u_cnt_y <= #1 out_u_cnt_y - (OD_BSIZE_MAX - 2);
          out_u_cnt_x <= #1 out_u_cnt_x + 'd2;
        end
      end
    end
  end
end

always_ff @(posedge clk) begin
  if (out_u_trn)
    out_u_lst <= #1 (out_u_cnt_y[5:0] == (OD_BSIZE_MAX - 2)) &&
        (out_u_cnt_x[5:0] == (OD_BSIZE_MAX - 2));
```

```
end

// out_v
assign out_v_trn = ((in_cnt_y > out_v_cnt_y) || (!in_rdy && |
    out_v_cnt_y)) && out_v_rdy;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_v_vld <= #1 'd0;
  else if (out_v_trn)
    out_v_vld <= #1 'd1;
  else if (out_v_rdy)
    out_v_vld <= #1 'd0;
end

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    out_v_cnt_x <= #1 'd0;
    out_v_cnt_y <= #1 'd0;
  end else if (out_v_trn) begin
    out_v_cnt_x <= #1 out_v_cnt_x + 'd2;

    if (out_v_cnt_x[5:0] == (OD_BSIZE_MAX - 2)) begin
      out_v_cnt_x <= #1 out_v_cnt_x - (OD_BSIZE_MAX - 2);
      out_v_cnt_y <= #1 out_v_cnt_y + 'd2;

      if (out_v_cnt_y[5:0] == (OD_BSIZE_MAX - 2)) begin
        if (out_v_cnt_x == w - 'd2) begin
          out_v_cnt_y <= #1 out_v_cnt_y + 'd2;
          out_v_cnt_x <= #1 'd0;
        end else begin
          out_v_cnt_y <= #1 out_v_cnt_y - (OD_BSIZE_MAX - 2);
          out_v_cnt_x <= #1 out_v_cnt_x + 'd2;
        end
      end
    end
  end
end

always_ff @(posedge clk) begin
  if (out_v_trn)
    out_v_lst <= #1 (out_v_cnt_y[5:0] == (OD_BSIZE_MAX - 2)) &&
        (out_v_cnt_x[5:0] == (OD_BSIZE_MAX - 2));
end

endmodule
```

```
import daala_pkg::*;

module haar_switch (
  input  logic        clk,
  input  logic        rst,

  input  logic        sof,
  input  logic  [4:0] nhsb,
  input  logic  [4:0] nvsb,

  input  logic        in_y_vld,
  input  logic  [7:0] in_y_dat,
  input  logic        in_y_lst,
  output logic        in_y_rdy,

  input  logic        in_u_vld,
  input  logic  [7:0] in_u_dat,
  input  logic        in_u_lst,
  output logic        in_u_rdy,

  input  logic        in_v_vld,
  input  logic  [7:0] in_v_dat,
  input  logic        in_v_lst,
  output logic        in_v_rdy,

  output logic        out_vld,
  output logic        out_lst,
  output logic  [7:0] out_dat,
  output logic  [2:0] out_ln,
  output logic  [1:0] out_pli,
  output logic  [4:0] out_sby,
  output logic  [4:0] out_sbx,
  input  logic        out_rdy
);

logic [1:0] in_sel;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    in_sel <= #1 'd0;
  else if (in_y_vld && in_y_rdy && in_y_lst)
    in_sel <= #1 'd1;
  else if (in_u_vld && in_u_rdy && in_u_lst)
    in_sel <= #1 'd2;
  else if (in_v_vld && in_v_rdy && in_v_lst)
    in_sel <= #1 'd0;
end
```

```
always_comb begin
  unique case(in_sel)
    'd0: begin
      out_vld = in_y_vld;
      out_dat = in_y_dat;
      out_lst = in_y_lst;
      out_ln = 'd6;
      out_pli = 'd0;
      in_y_rdy = out_rdy;
      in_u_rdy = 'd0;
      in_v_rdy = 'd0;
    end
    'd1: begin
      out_vld = in_u_vld;
      out_dat = in_u_dat;
      out_lst = in_u_lst;
      out_ln = 'd5;
      out_pli = 'd1;
      in_y_rdy = 'd0;
      in_u_rdy = out_rdy;
      in_v_rdy = 'd0;
    end
    'd2: begin
      out_vld = in_v_vld;
      out_dat = in_v_dat;
      out_lst = in_v_lst;
      out_ln = 'd5;
      out_pli = 'd2;
      in_y_rdy = 'd0;
      in_u_rdy = 'd0;
      in_v_rdy = out_rdy;
    end
  endcase
end

always_ff @(posedge clk) begin
  if (sof) begin
    out_sby <= #1 'd0;
    out_sbx <= #1 'd0;
  end else begin
    if (in_v_vld && in_v_rdy && in_v_lst) begin
      out_sbx <= #1 out_sbx + 'd1;
      if (out_sbx == nhsb - 1) begin
        out_sbx <= #1 'd0;
        out_sby <= #1 out_sby + 'd1;
      end
```

```
    end
  end
end


endmodule

import daala_pkg::*;

module haar (
  input  logic         clk,
  input  logic         rst,

  input  logic         in_vld,
  input  logic         in_lst,
  input  logic [2:0]   in_ln , // max 4+2
  input  logic [1:0]   in_pli,
  input  logic [4:0]   in_sbx,
  input  logic [4:0]   in_sby,
  input  logic [7:0]   in_x  ,
  output logic         in_rdy,

  output logic         sb_buff_en,
  output logic         sb_buff_wea,
  output logic         sb_buff_web,
  output logic [11:0]  sb_buff_adra,
  output logic [11:0]  sb_buff_adrb,
  output logic [15:0]  sb_buff_dina,
  output logic [15:0]  sb_buff_dinb,

  output logic         out_vld,
  output logic [OD_HAAR_BITS-1:0] out_dc,
  output logic [2:0]   out_ln,
  output logic [1:0]   out_pli,
  output logic [4:0]   out_sbx,
  output logic [4:0]   out_sby,
  input  logic         out_rdy
);

logic [OD_LOG_BSIZE_MAX-1:0] y;
logic [OD_LOG_BSIZE_MAX-1:0] x;
logic [OD_LOG_BSIZE_MAX-1:0] y_m2;
logic [OD_LOG_BSIZE_MAX-1:0] x_m2;
logic [OD_LOG_BSIZE_MAX-1:0] y_m2p1;
logic [OD_LOG_BSIZE_MAX  :0] n;
logic [OD_LOG_BSIZE_MAX-1:0] y_d1;
logic [OD_LOG_BSIZE_MAX-1:0] x_d1;
```

```
logic [OD_LOG_BSIZE_MAX-1:0] y_d1_pnp;
logic [OD_LOG_BSIZE_MAX-1:0] x_d1_pnp;
logic [2:0] ln;

logic [2:0] level;

logic  [OD_LOG_BSIZE_MAX-1:0] npairs;
logic  [OD_LOG_BSIZE_MAX-1:0] npairs_d1;

logic                         in_hk_vld;
logic                         hk_go;
logic  [OD_HAAR_BITS-1:0] in_hk_a;
logic  [OD_HAAR_BITS-1:0] in_hk_b;
logic  [OD_HAAR_BITS-1:0] in_hk_c;
logic  [OD_HAAR_BITS-1:0] in_hk_d;

logic                         out_hk_vld;
logic  [OD_HAAR_BITS-1:0] out_hk_a;
logic  [OD_HAAR_BITS-1:0] out_hk_b;
logic  [OD_HAAR_BITS-1:0] out_hk_c;
logic  [OD_HAAR_BITS-1:0] out_hk_d;

logic in_trn;
logic in_hk_lst;
logic out_hk_lst;

logic [31:0] tmp_ram_douta;
logic [31:0] tmp_ram_doutb;

dpram_sb_2048x32 tmp_ram_i (
  .clka   (clk),
  .ena    (in_trn || hk_go),
  .wea    (in_trn ? {{2{x[0]}} , {2{~x[0]}}}          : 4'b0000
     ),
  .addra  (in_trn ? {y,x[OD_LOG_BSIZE_MAX-1:1]}       : {y_m2,
     x_m2[OD_LOG_BSIZE_MAX-1:1]}),
  .dina   ({{8{in_x[7]}}, in_x, {8{in_x[7]}}, in_x}),
  .douta  (tmp_ram_douta),

  .clkb   (clk),
  .enb    (out_hk_vld || hk_go),
  .web    (out_hk_vld ? {{2{x_d1[0]}} , {2{~x_d1[0]}}}    : 4'
     b0000),
  .addrb  (out_hk_vld ? {y_d1,x_d1[OD_LOG_BSIZE_MAX-1:1]} : {
     y_m2p1, x_m2[OD_LOG_BSIZE_MAX-1:1]}),
  .dinb   ({{(16-OD_HAAR_BITS){1'd0}}, out_hk_a, {(16-
     OD_HAAR_BITS){1'd0}}, out_hk_a}),
```

```
  .doutb  (tmp_ram_doutb)
);

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    in_rdy <= #1 'd1;
  else if (in_vld && in_rdy && in_lst)
    in_rdy <= #1 'd0;
  else if (out_vld && out_rdy)
    in_rdy <= #1 'd1;
end

assign in_trn = in_vld && in_rdy;

assign in_hk_lst =  in_hk_vld && (x == npairs - 1) && (y ==
   npairs - 1) && (level == ln - 1);

assign n = 1 << ln;
assign npairs = n >> level >> 1;


always_ff @(posedge clk) begin
  if (rst) begin
    ln <= #1 'd6;
    level <= #1 'd0;
    y <= #1 'd0;
    x <= #1 'd0;
  end else if (in_trn) begin
    ln <= #1 in_ln;
    x <= #1 x + 'd1;
    if (x == n - 1) begin
      y <= #1 y + 'd1;
      x <= #1 'd0;
      if  (y == n - 1) begin
        y <= #1 'd0;
      end
    end
  end else if (in_hk_lst) begin
    level <= #1 'd0;
    y <= #1 'd0;
    x <= #1 'd0;
  end else if (in_hk_vld) begin
    x <= #1 x + 'd1;
    if (x == npairs - 1) begin
      y <= #1 y + 'd1;
      x <= #1 'd0;
      if  (y == npairs - 1) begin
```

```
        y <= #1 'd0;
        level <= #1 level + 'd1;
      end
    end
  end
end

assign y_m2 = {y, 1'b0};
assign x_m2 = {x, 1'b0};
assign y_m2p1 = {y, 1'b1};

assign in_hk_a = tmp_ram_douta[ 0+:OD_HAAR_BITS];
assign in_hk_c = tmp_ram_douta[16+:OD_HAAR_BITS];
assign in_hk_b = tmp_ram_doutb[ 0+:OD_HAAR_BITS];
assign in_hk_d = tmp_ram_doutb[16+:OD_HAAR_BITS];

always_ff @ (posedge clk, posedge rst) begin
  if (rst)
    hk_go <= #1 'd0;
  else if (in_trn && in_lst)
    hk_go <= #1 'd1;
  else if (in_hk_lst)
    hk_go <= #1 'd0;
end

always_ff @ (posedge clk, posedge rst) begin
  if (rst)
    in_hk_vld <= #1 'd0;
  else if (hk_go)
    in_hk_vld <= #1 ~in_hk_vld;
end

always_ff @(posedge clk) begin
  if (in_hk_vld) begin
    y_d1 <= #1 y;
    x_d1 <= #1 x;
    npairs_d1 <= #1 npairs;
    out_hk_lst <= #1 in_hk_lst;
  end
end
assign y_d1_pnp = y_d1 + npairs_d1;
assign x_d1_pnp = x_d1 + npairs_d1;

haar_kernel haar_kernel_i (
  .clk      (clk),
  .rst      (rst),
```

```
  .in_vld   (in_hk_vld ),
  .in_ll    (in_hk_a   ),
  .in_lh    (in_hk_b   ),
  .in_hl    (in_hk_c   ),
  .in_hh    (in_hk_d   ),

  .out_vld  (out_hk_vld),
  .out_ll   (out_hk_a  ),
  .out_lh   (out_hk_b  ),
  .out_hl   (out_hk_c  ),
  .out_hh   (out_hk_d  )
);


always_ff @(posedge clk) begin
    sb_buff_en <= #1 hk_go;
  if (hk_go) begin
    if (out_hk_vld) begin
      sb_buff_wea <= #1 'd1;
      sb_buff_web <= #1 'd1;
      sb_buff_adra <= #1 {y_d1    , x_d1_pnp};
      sb_buff_adrb <= #1 {y_d1_pnp, x_d1    };
      sb_buff_dina <= #1 out_hk_b;
      sb_buff_dinb <= #1 out_hk_c;
    end else begin
      sb_buff_wea <= #1 'd1;
      sb_buff_web <= #1 'd0;
      sb_buff_adra <= #1 {y_d1_pnp, x_d1_pnp};
      sb_buff_dina <= #1 out_hk_d;
    end
  end
end

always_ff @(posedge clk) begin
  if (out_hk_vld && out_hk_lst) begin
    out_dc <= #1 out_hk_a;
    out_pli <= #1 in_pli;
    out_sbx <= #1 in_sbx;
    out_sby <= #1 in_sby;
    out_ln <= #1 in_ln;
  end
end

always_ff @ (posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else if (out_hk_vld)
```

```
    out_vld <= #1 out_hk_lst;
  else if (out_rdy)
    out_vld <= #1 'd0;
end

endmodule

/*This is an in-place, reversible, orthonormal Haar transform
   in 7 adds,
   1 shift (2 operations per sample).
  It is its own inverse (but requires swapping lh and hl on one
      side for
   bit-exact reversibility).*/
import daala_pkg::*;

module haar_kernel (
  input  logic                    clk,
  input  logic                    rst,

  input  logic                    in_vld,
  input  logic [OD_HAAR_BITS-1:0] in_ll,
  input  logic [OD_HAAR_BITS-1:0] in_lh,
  input  logic [OD_HAAR_BITS-1:0] in_hl,
  input  logic [OD_HAAR_BITS-1:0] in_hh,

  output logic                     out_vld,
  output logic [OD_HAAR_BITS-1:0] out_ll,
  output logic [OD_HAAR_BITS-1:0] out_lh,
  output logic [OD_HAAR_BITS-1:0] out_hl,
  output logic [OD_HAAR_BITS-1:0] out_hh
);

logic [OD_HAAR_BITS-1:0] tmp_ll;
logic [OD_HAAR_BITS-1:0] tmp_lh;
logic [OD_HAAR_BITS-1:0] tmp_hl;
logic [OD_HAAR_BITS-1:0] tmp_hh;

logic [OD_HAAR_BITS-1:0] llmhh_2_diff;
logic [OD_HAAR_BITS-1:0] llmhh_2;

assign  tmp_ll = in_ll +  in_hl;
assign  tmp_hh = in_hh -  in_lh;

assign llmhh_2_diff = tmp_ll - tmp_hh;
assign llmhh_2 = {llmhh_2_diff[OD_HAAR_BITS-1], llmhh_2_diff[
   OD_HAAR_BITS-1:1]};
```

```
assign tmp_lh = llmhh_2 - in_lh;
assign tmp_hl = llmhh_2 - in_hl;


always_ff @(posedge clk) begin
  if (in_vld) begin
    out_ll <= #1 tmp_ll - tmp_lh;
    out_hh <= #1 tmp_hh + tmp_hl;
    out_lh <= #1 tmp_lh;
    out_hl <= #1 tmp_hl;
  end
end

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else
    out_vld <= #1 in_vld;
end

endmodule


import daala_pkg::*;

module haar_dc_sb (
  input  logic       clk,
  input  logic       rst,

  input  logic [4:0] nhsb,

  input  logic       in_vld,
  input  logic [OD_HAAR_BITS-1:0] in_dc,
  input  logic [2:0] in_ln,
  input  logic [1:0] in_pli,
  input  logic [4:0] in_sbx,
  input  logic [4:0] in_sby,

  output logic       quant_vld,
  output logic [OD_HAAR_BITS-1:0] quant_dc,
  output logic [1:0] quant_pli,
  input  logic       quant_rdy,

  output logic       out_vld,
  output logic [2:0] out_ln,
  output logic [1:0] out_pli,
  output logic [4:0] out_sbx,
  output logic [4:0] out_sby,
```

```
  input  logic       out_rdy
);
logic       in_vld_reg;
logic       in_vld_d1;

logic [2:0][31:0][31:0][OD_HAAR_BITS-1:0]sb_dc_mem;
logic [OD_HAAR_BITS-1:0]sb_dc_pred;

assign has_ur = (in_sby > 'd0) && (in_sbx < nhsb - 'd1);

always_ff @(posedge clk) begin
  if (in_vld) begin
    sb_dc_mem[in_pli][in_sby][in_sbx] <= #1 in_dc;
  end
end


always_ff @(posedge clk) begin
  if (in_vld) begin
    if (in_sby > 0 && in_sbx > 0) begin
      if (has_ur) begin
        sb_dc_pred = (22*sb_dc_mem[in_pli][in_sby][in_sbx - 1]
           -  9*sb_dc_mem[in_pli][in_sby - 1][in_sbx - 1] + 15*
           sb_dc_mem[in_pli][in_sby - 1][in_sbx] + 4*sb_dc_mem[
           in_pli][in_sby - 1][in_sbx + 1] + 16) >> 5;
      end else begin
        sb_dc_pred = (23*sb_dc_mem[in_pli][in_sby][in_sbx - 1]
           - 10*sb_dc_mem[in_pli][in_sby - 1][in_sbx - 1] + 19*
           sb_dc_mem[in_pli][in_sby - 1][in_sbx] + 16) >> 5;
      end
    end else if (in_sby > 0) begin
      sb_dc_pred = sb_dc_mem[in_pli][in_sby - 1][in_sbx];
    end else if (in_sbx > 0) begin
      sb_dc_pred = sb_dc_mem[in_pli][in_sby][in_sbx - 1];
    end else begin
      sb_dc_pred = 0;
    end
  end
end


always_ff @(posedge clk) begin
  if (in_vld_d1) begin
    quant_dc <= #1 in_dc - sb_dc_pred;
    quant_pli <= #1 in_pli;
    out_ln  <= #1 in_ln;
    out_pli <= #1 in_pli;
```

```
    out_sby <= #1 in_sby;
    out_sbx <= #1 in_sbx;
  end
end

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    in_vld_reg <= #1 'd0;
  end else if (in_vld && !in_vld_reg) begin
    in_vld_reg <= #1 'd1;
  end else if (out_vld && out_rdy) begin
    in_vld_reg <= #1 'd0;
  end
end



always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    in_vld_d1 <= #1 'd0;
  end else begin
    in_vld_d1 <= #1 in_vld && !in_vld_reg;
  end
end


always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    quant_vld <= #1 'd0;
  end else if (in_vld_d1) begin
    quant_vld <= #1 'd1;
  end else if (quant_rdy) begin
    quant_vld <= #1 'd0;
  end
end


always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    out_vld <= #1 'd0;
  end else if (quant_vld && quant_rdy) begin
    out_vld <= #1 'd1;
  end else if (out_rdy) begin
    out_vld <= #1 'd0;
  end
end
```

```
endmodule


import daala_pkg::*;

module max_tree (
  input  logic        clk,
  input  logic        rst,

  input  logic        in_vld,
  input  logic [2:0]  in_ln,
  input  logic [1:0]  in_pli,
  output logic        in_rdy,

  output logic         sb_coef_buff_en,
  output logic [11:0]  sb_coef_buff_adr,
  input  logic [15:0]  sb_coef_buff_dout,

  output logic         sb_tree_sum_buff_en,
  output logic  [1:0]  sb_tree_sum_buff_wea,
  output logic [10:0]  sb_tree_sum_buff_adra,
  output logic [10:0]  sb_tree_sum_buff_adrb,
  output logic [31:0]  sb_tree_sum_buff_dina,
  input  logic [31:0]  sb_tree_sum_buff_douta,
  input  logic [31:0]  sb_tree_sum_buff_doutb,

  output logic        out_vld,
  output logic [2:0]  out_ln,
  output logic [1:0]  out_pli,
  input  logic        out_rdy
);


logic [OD_LOG_BSIZE_MAX-1:0] y;
logic [OD_LOG_BSIZE_MAX-1:0] x;
logic [OD_LOG_BSIZE_MAX  :0] n;
logic [OD_LOG_BSIZE_MAX-1:0] y_m2;
logic [OD_LOG_BSIZE_MAX-1:0] x_m2;
logic [OD_LOG_BSIZE_MAX-1:0] y_m2p1;
logic [OD_LOG_BSIZE_MAX-1:0] x_m2p1;
logic                        y_eq_0;
logic                        x_eq_0;
logic                        y_gte_n2;
logic                        x_gte_n2;

logic [OD_HAAR_BITS-1:0] abs_c;

logic in_vld_stb;
```

```
logic in_vld_stb_reg;
logic in_lst;

logic tree_sum_write;
logic [1:0][1:0][OD_MAX_TREE_BITS-1:0] tree_sum;

assign in_lst = in_vld_stb_reg && x_eq_0 && y_eq_0;

assign in_vld_stb = in_vld && !in_vld_stb_reg;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    in_vld_stb_reg <= #1 'd0;
  else if (in_vld_stb)
    in_vld_stb_reg <= #1 'd1;
  else if (in_lst)
    in_vld_stb_reg <= #1 'd0;
end

assign in_rdy = !in_vld || (out_rdy && in_lst);

assign n = 1 << in_ln;

always_ff @(posedge clk) begin
  if (rst) begin
    y <= #1 'd0;
    x <= #1 'd0;
  end else if (in_vld_stb) begin
    y <= #1 n - 'd1;
    x <= #1 n - 'd1;
  end else if (!in_lst && tree_sum_write) begin
    x <= #1 x - 'd1;
    if (x == 'd0) begin
      y <= #1 y - 'd1;
      x <= #1 n - 'd1;
    end
  end
end

assign x_m2 = {x, 1'b0};
assign y_m2 = {y, 1'b0};
assign x_m2p1 = {x, 1'd1};
assign y_m2p1 = {y, 1'd1};
assign x_eq_0 = x == 'd0;
assign y_eq_0 = y == 'd0;
assign x_gte_n2 = x >= (n>>1);
assign y_gte_n2 = y >= (n>>1);
```

```
assign sb_coef_buff_en = 1'b1;
assign sb_coef_buff_adr = {y, x};
assign abs_c = sb_coef_buff_dout[OD_HAAR_BITS-1] ? -
   sb_coef_buff_dout[OD_HAAR_BITS-1:0] : sb_coef_buff_dout[
   OD_HAAR_BITS-1:0];

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else if (in_lst)
    out_vld <= #1 'd1;
  else if (out_rdy)
    out_vld <= #1 'd0;
end

always_ff @(posedge clk) begin
  if (in_vld_stb)
    tree_sum_write <= #1 'd0;
  else
    tree_sum_write <= #1 ~tree_sum_write;
end


always_ff @(posedge clk) begin
  if (in_vld) begin
      sb_tree_sum_buff_en      <= #1 'd1;
    if (tree_sum_write) begin
      sb_tree_sum_buff_wea <= #1 {x[0], ~x[0]};
      sb_tree_sum_buff_adra <= #1 {y,x[OD_LOG_BSIZE_MAX-1:1]};
      sb_tree_sum_buff_dina <= #1 abs_c + ((x_gte_n2 ||
         y_gte_n2) ? 'd0 : (in_lst ? 'd0 : tree_sum[0][0]) +
         tree_sum[0][1] + tree_sum[1][0] + tree_sum[1][1]);
    end else begin
      sb_tree_sum_buff_wea <= #1 2'b00;
      sb_tree_sum_buff_adra <= #1 {y_m2  ,x_m2[OD_LOG_BSIZE_MAX
         -1:1]};
      sb_tree_sum_buff_adrb <= #1 {y_m2p1,x_m2[OD_LOG_BSIZE_MAX
         -1:1]};
    end
  end
end
assign tree_sum[0][0] = sb_tree_sum_buff_douta[ 0+:
   OD_MAX_TREE_BITS];
assign tree_sum[0][1] = sb_tree_sum_buff_douta[16+:
   OD_MAX_TREE_BITS];
assign tree_sum[1][0] = sb_tree_sum_buff_doutb[ 0+:
```

```
   OD_MAX_TREE_BITS];
assign tree_sum[1][1] = sb_tree_sum_buff_doutb[16+:
   OD_MAX_TREE_BITS];

always_ff @(posedge clk) begin
  if (in_vld) begin
    if (in_lst) begin
      out_ln <= #1 in_ln;
      out_pli <= #1 in_pli;
    end
  end
end

endmodule


import daala_pkg::*;

module max_tree_entenc (
  input  logic       clk,
  input  logic       rst,

  input  logic        sof,
  input  logic  [4:0] nhsb,
  input  logic  [4:0] nvsb,
  output logic        eob,
  output logic        eof,

  input  logic        in_vld,
  input  logic [ 2:0] in_ln,
  input  logic [ 1:0] in_pli,
  output logic        in_rdy,

  output logic        sb_coef_buff_en,
  output logic [11:0] sb_coef_buff_adr,
  input  logic [15:0] sb_coef_buff_dout,

  output logic        sb_tree_sum_buff_en,
  output logic [10:0] sb_tree_sum_buff_adra,
  output logic [10:0] sb_tree_sum_buff_adrb,
  input  logic [31:0] sb_tree_sum_buff_douta,
  input  logic [31:0] sb_tree_sum_buff_doutb,

  output logic        out_vld,
  output logic [ 3:0] out_val,
  output logic [ 1:0] out_cdf_sel,
  output logic [ 8:0] out_cdf_num,
  output logic [ 4:0] out_nsyms,
```

```
  input  logic        out_rdy,

  output logic        bits_vld,
  output logic [31:0] bits_fl ,
  output logic [ 4:0] bits_ftb,
  input  logic        bits_rdy
);

localparam  [3:0] STATE_INIT            = 'h0;
localparam  [3:0] STATE_BITS_MIN        = 'h1;
localparam  [3:0] STATE_BITS_GT_15      = 'h2;
localparam  [3:0] STATE_BITS_GTE_15     = 'h3;
localparam  [3:0] STATE_BITS_GT_1       = 'h4;
localparam  [3:0] STATE_ENC_TOP_TS_DIAG = 'h5;
localparam  [3:0] STATE_ENC_TOP_TS_HV   = 'h6;
localparam  [3:0] STATE_ENC_TREE        = 'h7;
localparam  [3:0] STATE_ENC_SIGN        = 'h8;

logic [3:0] state;

logic [OD_LOG_BSIZE_MAX:0] n;
logic [5:0] bits;


logic [31:0] split_a;
logic [31:0] split_sum;
logic        split_sum_neq_0;
logic [ 3:0] split_ctx;
logic [ 4:0] split_shift;
logic        split_shift_neq_0;

localparam [2:0]  STATE_TREE_CHECK   = 'd0;
localparam [2:0]  STATE_TREE_COEF    = 'd1;
localparam [2:0]  STATE_TREE_SPLIT_0 = 'd2;
localparam [2:0]  STATE_TREE_SPLIT_1 = 'd3;
localparam [2:0]  STATE_TREE_SPLIT_2 = 'd4;

logic        tree_next;
logic [ 2:0] tree_state;
logic        tree_sum_neq_0;
logic [31:0] tree_children_sum;
logic        tree_children_sum_neq_0;
logic [ 3:0] tree_dir_calc;
logic [ 1:0] tree_dir;
logic        tree_dir_eq_0;

logic [31:0] tree_split_a;
```

```
logic [31:0] tree_split_sum;
logic [ 3:0] tree_split_ctx;

logic                          tree_next_m2;
logic [OD_LOG_BSIZE_MAX-2:0] tree_y;
logic [OD_LOG_BSIZE_MAX-2:0] tree_x;
logic [OD_LOG_BSIZE_MAX-1:0] tree_y_m2;
logic [OD_LOG_BSIZE_MAX-1:0] tree_x_m2;
logic [OD_LOG_BSIZE_MAX-1:0] tree_y_m2p1;
logic [OD_LOG_BSIZE_MAX-2:0] tree_y_next;
logic [OD_LOG_BSIZE_MAX-2:0] tree_x_next;
logic                          tree_y_lt_n_d4;
logic                          tree_x_lt_n_d4;
logic [OD_LOG_BSIZE_MAX-2:0] tree_xy_max;

logic [OD_LOG_BSIZE_MAX-2:0] tree_cnt_and;
logic                  [ 1:0] tree_cnt_end;
logic                  [ 1:0] tree_cnt_end_next;
logic                  [ 2:0] tree_cnt_shift;
logic                  [ 2:0] tree_cnt_shift_idx;

logic [OD_LOG_BSIZE_MAX-1:0] sign_y;
logic [OD_LOG_BSIZE_MAX-1:0] sign_x;
logic                          sign_c_neq_0;

logic [1:0][1:0][OD_MAX_TREE_BITS-1:0] in_tree_sum;

// coef buff
always_comb begin
  case (state)
    STATE_ENC_SIGN        : begin
      sb_coef_buff_en = 'd1;
      sb_coef_buff_adr = {sign_y, sign_x};
    end
    STATE_ENC_TREE        : begin
      sb_coef_buff_en = 'd1;
      sb_coef_buff_adr = {tree_y, tree_x};
    end
    default         : begin
      sb_coef_buff_en = 'd0;
      sb_coef_buff_adr = 'dx;
    end
  endcase
end

// tree_sum buff
always_comb begin
```

```
  case (state)
    STATE_BITS_MIN,
    STATE_BITS_GT_15,
    STATE_BITS_GTE_15,
    STATE_BITS_GT_1,
    STATE_ENC_TOP_TS_DIAG ,
    STATE_ENC_TOP_TS_HV   : begin
      sb_tree_sum_buff_en = 'd1;
      sb_tree_sum_buff_adra = 'd0;
      sb_tree_sum_buff_adrb = {{OD_LOG_BSIZE_MAX-1{1'b0}}, 1'd1
          , {OD_LOG_BSIZE_MAX-1{1'b0}}};
    end
    STATE_ENC_TREE        : begin
      sb_tree_sum_buff_en = 'd1;
      sb_tree_sum_buff_adra = {tree_y_m2  , tree_x_m2[
          OD_LOG_BSIZE_MAX-1:1]};
      sb_tree_sum_buff_adrb = {tree_y_m2p1, tree_x_m2[
          OD_LOG_BSIZE_MAX-1:1]};
    end
    default        : begin
      sb_tree_sum_buff_en = 'd0;
      sb_tree_sum_buff_adra = 'dx;
      sb_tree_sum_buff_adrb = 'dx;
    end
  endcase
end
assign in_tree_sum[0][0] = sb_tree_sum_buff_douta[ 0+:
   OD_MAX_TREE_BITS];
assign in_tree_sum[0][1] = sb_tree_sum_buff_douta[16+:
   OD_MAX_TREE_BITS];
assign in_tree_sum[1][0] = sb_tree_sum_buff_doutb[ 0+:
   OD_MAX_TREE_BITS];
assign in_tree_sum[1][1] = sb_tree_sum_buff_doutb[16+:
   OD_MAX_TREE_BITS];

// state fsm
assign n = 1 << in_ln;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    state <= #1 'd0;
    else if (out_rdy && bits_rdy) begin
    case (state)
      STATE_INIT            : state <= #1 in_vld ?
          STATE_BITS_MIN : STATE_INIT;
      STATE_BITS_MIN        : state <= #1((bits >  'd15) ?
          STATE_BITS_GT_15 :
```

```
                              ((bits >= 'd15) ?
                                 STATE_BITS_GTE_15:
                              ((bits >   'd1) ? STATE_BITS_GT_1
                                   :
                              STATE_ENC_TOP_TS_DIAG)));
      STATE_BITS_GT_15      : state <= #1((bits >= 'd15) ?
         STATE_BITS_GTE_15:
                              ((bits >   'd1) ? STATE_BITS_GT_1
                                   :
                              STATE_ENC_TOP_TS_DIAG));

      STATE_BITS_GTE_15     : state <= #1((bits >   'd1) ?
         STATE_BITS_GT_1  :
                              STATE_ENC_TOP_TS_DIAG);
      STATE_BITS_GT_1       : state <= #1 STATE_ENC_TOP_TS_DIAG
         ;
      STATE_ENC_TOP_TS_DIAG : state <= #1 STATE_ENC_TOP_TS_HV;
      STATE_ENC_TOP_TS_HV   : state <= #1 STATE_ENC_TREE;
      STATE_ENC_TREE        : state <= #1 tree_lst ?
         STATE_ENC_SIGN : STATE_ENC_TREE;
      STATE_ENC_SIGN        : state <= #1 sign_lst ? STATE_INIT
          : STATE_ENC_SIGN;
      default               : state <= #1 STATE_INIT;
    endcase
  end
end

always_comb begin
  case (1'b1)
    in_tree_sum[0][0][15]: bits = 'd16;
    in_tree_sum[0][0][14]: bits = 'd15;
    in_tree_sum[0][0][13]: bits = 'd14;
    in_tree_sum[0][0][12]: bits = 'd13;
    in_tree_sum[0][0][11]: bits = 'd12;
    in_tree_sum[0][0][10]: bits = 'd11;
    in_tree_sum[0][0][09]: bits = 'd10;
    in_tree_sum[0][0][08]: bits = 'd09;
    in_tree_sum[0][0][07]: bits = 'd08;
    in_tree_sum[0][0][06]: bits = 'd07;
    in_tree_sum[0][0][05]: bits = 'd06;
    in_tree_sum[0][0][04]: bits = 'd05;
    in_tree_sum[0][0][03]: bits = 'd04;
    in_tree_sum[0][0][02]: bits = 'd03;
    in_tree_sum[0][0][01]: bits = 'd02;
    in_tree_sum[0][0][00]: bits = 'd01;
    default              : bits = 'd00;
  endcase
```

```
end

// cdf adapt
always_comb begin
  case(state)
    STATE_BITS_MIN         : out_vld = 'd1;
    STATE_ENC_TOP_TS_DIAG ,
    STATE_ENC_TOP_TS_HV    : out_vld = split_sum_neq_0;
    STATE_ENC_TREE         :
      case (tree_state)
        STATE_TREE_CHECK   : out_vld = 'd0;
        default            : out_vld = split_sum_neq_0;
      endcase
    default                : out_vld = 'd0;
  endcase
end

always_comb begin
  case(state)
    STATE_BITS_MIN         : out_val = daala_min_6(bits, 'd15);
    STATE_ENC_TOP_TS_DIAG ,
    STATE_ENC_TOP_TS_HV    : out_val = split_a >> split_shift;
    default                : out_val = 'd0;
  endcase
end

always_comb begin
  case(state)
    STATE_BITS_MIN         : out_cdf_sel = 'd2;
    STATE_ENC_TOP_TS_DIAG ,
    STATE_ENC_TOP_TS_HV    : out_cdf_sel = 'd1;
    default                :
      case (tree_state)
        STATE_TREE_COEF    : out_cdf_sel = 'd0;
        default            : out_cdf_sel = 'd1;
      endcase
  endcase
end

always_comb begin
  case(state)
    STATE_BITS_MIN         : out_cdf_num = in_pli;
    STATE_ENC_TOP_TS_DIAG ,
    STATE_ENC_TOP_TS_HV    : out_cdf_num = 'd15*('d2*split_ctx +
        daala_min_5(split_shift, 5'd1)) + (split_sum >>
       split_shift) - 'd1;
    default                :
```

```
      case (tree_state)
        STATE_TREE_COEF   : out_cdf_num = 'd15*split_ctx +
           split_sum - 'd1;
        default           : out_cdf_num = 'd15*(2*split_ctx +
           daala_min_5(split_shift, 1)) + split_sum - 1;
      endcase
  endcase
end

always_comb begin
  case(state)
    STATE_ENC_TREE,
    STATE_ENC_TOP_TS_DIAG ,
    STATE_ENC_TOP_TS_HV   : out_nsyms = (split_sum >>
       split_shift) + 'd1;
    default               : out_nsyms = 'd16;
  endcase
end


// bits
always_comb begin
  case(state)
    STATE_BITS_GT_15      ,
    STATE_BITS_GTE_15     ,
    STATE_BITS_GT_1       : bits_vld = 'd1;
    STATE_ENC_TOP_TS_DIAG ,
    STATE_ENC_TOP_TS_HV   : bits_vld = split_sum_neq_0 &&
       split_shift_neq_0;
    STATE_ENC_SIGN        : bits_vld = sign_c_neq_0;
    STATE_ENC_TREE        :
      case (tree_state)
        STATE_TREE_CHECK  : bits_vld = 'd0;
        default           : bits_vld = split_sum_neq_0 &&
           split_shift_neq_0;
      endcase
    default               : bits_vld = 'd0;
  endcase
end

always_comb begin
  case(state)
    STATE_BITS_GT_15      : bits_fl = 'd0;
    STATE_BITS_GTE_15     : bits_fl = 'd1;
    STATE_BITS_GT_1       : bits_fl = in_tree_sum[0][0] & ((1
       << (bits - 1)) - 1);
    STATE_ENC_TREE,
```

```
    STATE_ENC_TOP_TS_DIAG ,
    STATE_ENC_TOP_TS_HV   : bits_fl = split_a & ((1 <<
       split_shift) - 1);
    STATE_ENC_SIGN        : bits_fl = sb_coef_buff_dout[
       OD_HAAR_BITS-1];
    default               : bits_fl = 'dx;
  endcase
end

always_comb begin
  case(state)
    STATE_BITS_GT_15     : bits_ftb = bits - 'd15;
    STATE_BITS_GTE_15    : bits_ftb = 'd1;
    STATE_BITS_GT_1      : bits_ftb = bits -1;
    STATE_ENC_TREE,
    STATE_ENC_TOP_TS_DIAG,
    STATE_ENC_TOP_TS_HV  : bits_ftb = split_shift;
    STATE_ENC_SIGN       : bits_ftb = 'd1;
    default              : bits_ftb = 'dx;
  endcase
end

// tree split
always_comb begin
  case(state)
    STATE_ENC_TOP_TS_DIAG: begin
      split_a   = in_tree_sum[1][1];
      split_sum = in_tree_sum[0][0];
      split_ctx = 'd3;
    end
    STATE_ENC_TOP_TS_HV  : begin
      split_a   = in_tree_sum[0][1];
      split_sum = in_tree_sum[0][0] - in_tree_sum[1][1];
      split_ctx = 'd4;
    end
    default              : begin
      split_a   = tree_split_a;
      split_sum = tree_split_sum;
      split_ctx = tree_split_ctx;
    end
  endcase
end

assign split_sum_neq_0 = split_sum != 'd0;

always_comb begin
  case (1'b1)
```

```
    split_sum[31]: split_shift = 'd28;
    split_sum[30]: split_shift = 'd27;
    split_sum[29]: split_shift = 'd26;
    split_sum[28]: split_shift = 'd25;
    split_sum[27]: split_shift = 'd24;
    split_sum[26]: split_shift = 'd23;
    split_sum[25]: split_shift = 'd22;
    split_sum[24]: split_shift = 'd21;
    split_sum[23]: split_shift = 'd20;
    split_sum[22]: split_shift = 'd19;
    split_sum[21]: split_shift = 'd18;
    split_sum[20]: split_shift = 'd17;
    split_sum[19]: split_shift = 'd16;
    split_sum[18]: split_shift = 'd15;
    split_sum[17]: split_shift = 'd14;
    split_sum[16]: split_shift = 'd13;
    split_sum[15]: split_shift = 'd12;
    split_sum[14]: split_shift = 'd11;
    split_sum[13]: split_shift = 'd10;
    split_sum[12]: split_shift = 'd09;
    split_sum[11]: split_shift = 'd08;
    split_sum[10]: split_shift = 'd07;
    split_sum[09]: split_shift = 'd06;
    split_sum[08]: split_shift = 'd05;
    split_sum[07]: split_shift = 'd04;
    split_sum[06]: split_shift = 'd03;
    split_sum[05]: split_shift = 'd02;
    split_sum[04]: split_shift = 'd01;
    default      : split_shift = 'd00;
  endcase
end

assign split_shift_neq_0 = split_shift != 'd0;

// tree encode
always_ff @(posedge clk) begin
  if (state == STATE_ENC_TOP_TS_HV) begin
    tree_state <= #1 STATE_TREE_CHECK;
  end else if (out_rdy && bits_rdy) begin
    unique case (tree_state)
      STATE_TREE_CHECK  : tree_state <= #1 tree_sum_neq_0 ?
         STATE_TREE_COEF : STATE_TREE_CHECK;
      STATE_TREE_COEF   : tree_state <= #1
         tree_children_sum_neq_0 ? STATE_TREE_SPLIT_0 :
         STATE_TREE_CHECK;
      STATE_TREE_SPLIT_0: tree_state <= #1 STATE_TREE_SPLIT_1;
      STATE_TREE_SPLIT_1: tree_state <= #1 STATE_TREE_SPLIT_2;
```

```
      STATE_TREE_SPLIT_2: tree_state <= #1 STATE_TREE_CHECK;
    endcase
  end
end

assign tree_next = ((tree_state == STATE_TREE_CHECK) && !
   tree_sum_neq_0) || ((tree_state == STATE_TREE_COEF) && !
   tree_children_sum_neq_0) || (tree_state ==
   STATE_TREE_SPLIT_2);

assign tree_sum_neq_0 = in_tree_sum[tree_y][tree_x] != 'd0;

always_ff @(posedge clk) begin
  if (state == STATE_ENC_TOP_TS_HV)
    tree_dir <= #1 'd0;
  else if (out_rdy && bits_rdy && tree_next && (state ==
     STATE_ENC_TREE)) begin
    if ((tree_y_next < 'd2) && (tree_x_next < 'd2) && !
       tree_next_m2) begin
      tree_dir <= #1 tree_dir + 'd1;
    end
  end
end
assign tree_dir_eq_0 = tree_dir == 'd0;

assign tree_xy_max = daala_max_5(tree_x, tree_y);
always_comb begin
  case (1'b1)
    tree_xy_max[4]: tree_dir_calc = 'd12;
    tree_xy_max[3]: tree_dir_calc = 'd9;
    tree_xy_max[2]: tree_dir_calc = 'd6;
    tree_xy_max[1]: tree_dir_calc = 'd3;
    default       : tree_dir_calc = 'd0;
  endcase
end


always_ff @(posedge clk) begin
  if (out_rdy && bits_rdy) begin
    unique case (tree_state)
      STATE_TREE_CHECK: begin
        tree_split_a    <= #1 sb_coef_buff_dout[OD_HAAR_BITS-1]
            ? - sb_coef_buff_dout[OD_HAAR_BITS-1:0] :
           sb_coef_buff_dout[OD_HAAR_BITS-1:0];
        tree_split_sum  <= #1 in_tree_sum[tree_y][tree_x];
        tree_split_ctx  <= #1 tree_dir + tree_dir_calc;
```

```
        tree_children_sum <= #1 in_tree_sum[0][0] + in_tree_sum
            [0][1] + in_tree_sum[1][0] + in_tree_sum[1][1];
      end
      STATE_TREE_COEF: begin
        tree_split_a    <= #1 tree_dir_eq_0 ? in_tree_sum[0][0]
             + in_tree_sum[0][1] : in_tree_sum[0][0] +
            in_tree_sum[1][0];
        tree_split_sum  <= #1 tree_children_sum;
        tree_split_ctx  <= #1 tree_dir_eq_0 ? 'd0 : 'd1;
      end
      STATE_TREE_SPLIT_0: begin
        tree_split_a    <= #1 in_tree_sum[0][0];
        tree_split_sum  <= #1 tree_dir_eq_0 ? in_tree_sum[0][0]
             + in_tree_sum[0][1] : in_tree_sum[0][0] +
            in_tree_sum[1][0];
        tree_split_ctx  <= #1 'd2;
      end
      STATE_TREE_SPLIT_1: begin
        tree_split_a    <= #1 tree_dir_eq_0 ? in_tree_sum[1][0]
                              : in_tree_sum[0][1];
        tree_split_sum  <= #1 tree_dir_eq_0 ? in_tree_sum[1][0]
             + in_tree_sum[1][1] : in_tree_sum[0][1] +
            in_tree_sum[1][1];
        tree_split_ctx  <= #1 'd2;
      end
    endcase
  end
end
assign tree_children_sum_neq_0 = tree_children_sum != 'd0;

// tree encode step logic
assign tree_next_m2 = (tree_sum_neq_0 && tree_y_lt_n_d4 &&
   tree_x_lt_n_d4);
always_ff @(posedge clk) begin
  if (state == STATE_ENC_TOP_TS_HV) begin
    tree_y <= #1 'd0;
    tree_x <= #1 'd1;
  end else if (out_rdy && bits_rdy && tree_next) begin
    if (tree_next_m2) begin
      tree_y <= #1 tree_y << 1;
      tree_x <= #1 tree_x << 1;
    end else begin
      tree_y <= #1 tree_y_next;
      tree_x <= #1 tree_x_next;
    end
  end
end
```

```
assign tree_y_lt_n_d4 = tree_y < (n >> 2);
assign tree_x_lt_n_d4 = tree_x < (n >> 2);

assign tree_y_m2 = {tree_y, 1'd0};
assign tree_x_m2 = {tree_x, 1'd0};
assign tree_y_m2p1 = {tree_y, 1'd1};

assign tree_cnt_and = tree_y & tree_x;

assign tree_cnt_end = {tree_y[tree_cnt_shift_idx], tree_x[
   tree_cnt_shift_idx]};
assign tree_cnt_end_next = tree_cnt_end + 'd1;
assign tree_y_next = {tree_y >> tree_cnt_shift,
   tree_cnt_end_next[1]};
assign tree_x_next = {tree_x >> tree_cnt_shift,
   tree_cnt_end_next[0]};

always_comb begin
  case (1'b1)
    &tree_cnt_and[3:0]: tree_cnt_shift = 'd5;
    &tree_cnt_and[2:0]: tree_cnt_shift = 'd4;
    &tree_cnt_and[1:0]: tree_cnt_shift = 'd3;
    &tree_cnt_and[0:0]: tree_cnt_shift = 'd2;
    default           : tree_cnt_shift = 'd1;
  endcase
end
always_comb begin
  case (1'b1)
    &tree_cnt_and[3:0]: tree_cnt_shift_idx = 'd4;
    &tree_cnt_and[2:0]: tree_cnt_shift_idx = 'd3;
    &tree_cnt_and[1:0]: tree_cnt_shift_idx = 'd2;
    &tree_cnt_and[0:0]: tree_cnt_shift_idx = 'd1;
    default           : tree_cnt_shift_idx = 'd0;
  endcase
end

assign tree_lst = !tree_next_m2 && (tree_dir == 'd2) && (
   tree_y_next < 'd2) && (tree_x_next < 'd2);

// sign
always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    sign_y <= #1 'd0;
    sign_x <= #1 'd1;
  end else if ((state == STATE_ENC_SIGN) && bits_rdy) begin
    if (sign_lst) begin
```

```
      sign_y <= #1 'd0;
      sign_x <= #1 'd1;
    end else begin
      sign_x <= #1 sign_x + 'd1;
      if (sign_x == n - 'd1) begin
        sign_x <= #1 'd0;
        sign_y <= #1 sign_y + 'd1;
      end
    end
  end
end

assign sign_lst = (sign_x == n - 'd1) && (sign_y == n - 'd1);

assign sign_c_neq_0 = sb_coef_buff_dout[OD_HAAR_BITS-1:0] != '
   d0;

assign in_rdy = (state == STATE_ENC_SIGN) && sign_lst &&
   bits_rdy;

// end of frame
logic [4:0] in_sbx;
logic [4:0] in_sby;

always_ff @(posedge rst, posedge clk) begin
  if (rst) begin
    in_sby <= #1 'd0;
    in_sbx <= #1 'd0;
  end else begin
    if (eof) begin
      in_sby <= #1 'd0;
      in_sbx <= #1 'd0;
    end else if (in_vld && in_rdy) begin
      in_sbx <= #1 in_sbx + 'd1;
      if (in_sbx == nhsb - 1) begin
        in_sbx <= #1 'd0;
        in_sby <= #1 in_sby + 'd1;
      end
    end
  end
end

assign eof = in_rdy && ((in_pli == 'd2) && (in_sbx == nhsb - 1)
    && (in_sby == nvsb - 1));
assign eob = in_rdy;

endmodule
```

```
import daala_pkg::*;

module gen_enc (
  input  logic        rst,
  input  logic        clk,

  input  logic        sof,

  input  logic        in_vld,
  input  logic [OD_HAAR_BITS-1:0] in_x,
  input  logic [ 1:0] in_pli,
  output logic        in_rdy,

  output logic        out_vld,
  output logic [15:0] out_fl,
  output logic [15:0] out_fh,
  output logic [15:0] out_ft,
  input  logic        out_rdy,

  output logic        laplace_vld,
  output logic [15:0] laplace_x,
  output logic [15:0] laplace_decay,
  input  logic        laplace_rdy,

  output logic        bits_vld,
  output logic [31:0] bits_ft,
  output logic [ 4:0] bits_ftb,
  input  logic        bits_rdy
);

logic [31:0] abs_x;

logic        in_vld_reg;
logic        in_vld_d1;
logic        in_vld_d2;
logic        bits_vld_x;
logic        bits_vld_s;

logic [2:0][31:0] ex_q16;
logic      [31:0] ex_q16_curr;
logic [15:0] e;
logic [15:0] e_calc;
logic [15:0] e_calc_d1;

logic [2:0][11:0][15:0][15:0] dc_cdf;
logic      [15:0][15:0] dc_cdf_curr;
```

```
logic             dc_cdf_adapt;

logic [ 3:0] id;
logic [ 3:0] shift;
logic        shift_gt_spec;
logic [15:0] xs;
logic        xs_gte_15;
logic        special;
logic [ 5:0] lg;
logic [ 5:0] tmp;
logic [ 5:0] odd;
logic [ 4:0] lg_q1;

logic [15:0] ft_m1;
logic [ 3:0] s;
logic        s_gt_0;
logic [ 3:0] s2;

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    in_vld_reg <= #1 'd0;
  end else if (in_vld && !in_vld_reg) begin
    in_vld_reg <= #1 'd1;
  end else if (in_rdy) begin
    in_vld_reg <= #1 'd0;
  end
end
 always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    in_vld_d1 <= #1 'd0;
    in_vld_d2 <= #1 'd0;
  end else begin
    in_vld_d1 <= #1 in_vld && !in_vld_reg;
    in_vld_d2 <= #1 in_vld_d1;
  end
end

// update state
assign dc_cdf_adapt = dc_cdf_curr[15] > 'd32767 -
   OD_ADAPT_DC_INCREMENT;
assign xenc = s;

always_ff @(posedge clk) begin
  if (sof) begin
    dc_cdf[0] <= #1 {12{16'd1024, 16'd960, 16'd896, 16'd832,
       16'd768, 16'd704, 16'd640, 16'd576, 16'd512, 16'd448,
       16'd384, 16'd320, 16'd256, 16'd192, 16'd128, 16'd64}};
```

```
  dc_cdf[1] <= #1 {12{16'd1024, 16'd960, 16'd896, 16'd832,
     16'd768, 16'd704, 16'd640, 16'd576, 16'd512, 16'd448,
     16'd384, 16'd320, 16'd256, 16'd192, 16'd128, 16'd64}};
  dc_cdf[2] <= #1 {12{16'd1024, 16'd960, 16'd896, 16'd832,
     16'd768, 16'd704, 16'd640, 16'd576, 16'd512, 16'd448,
     16'd384, 16'd320, 16'd256, 16'd192, 16'd128, 16'd64}};
end else if (in_vld && in_rdy) begin
  dc_cdf[in_pli][id][15] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][15][15:1] + 'd16 : dc_cdf[in_pli][id][15]) + ((
     xenc >= 15) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][14] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][14][15:1] + 'd15 : dc_cdf[in_pli][id][14]) + ((
     xenc >= 14) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][13] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][13][15:1] + 'd14 : dc_cdf[in_pli][id][13]) + ((
     xenc >= 13) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][12] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][12][15:1] + 'd13 : dc_cdf[in_pli][id][12]) + ((
     xenc >= 12) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][11] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][11][15:1] + 'd12 : dc_cdf[in_pli][id][11]) + ((
     xenc >= 11) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][10] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][10][15:1] + 'd11 : dc_cdf[in_pli][id][10]) + ((
     xenc >= 10) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][09] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][09][15:1] + 'd10 : dc_cdf[in_pli][id][09]) + ((
     xenc >= 09) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][08] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][08][15:1] + 'd09 : dc_cdf[in_pli][id][08]) + ((
     xenc >= 08) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][07] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][07][15:1] + 'd08 : dc_cdf[in_pli][id][07]) + ((
     xenc >= 07) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][06] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][06][15:1] + 'd07 : dc_cdf[in_pli][id][06]) + ((
     xenc >= 06) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][05] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][05][15:1] + 'd06 : dc_cdf[in_pli][id][05]) + ((
     xenc >= 05) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][04] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][04][15:1] + 'd05 : dc_cdf[in_pli][id][04]) + ((
     xenc >= 04) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][03] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
     ][id][03][15:1] + 'd04 : dc_cdf[in_pli][id][03]) + ((
     xenc >= 03) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  dc_cdf[in_pli][id][02] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
```

```
        ][id][02][15:1] + 'd03 : dc_cdf[in_pli][id][02]) + ((
        xenc >= 02) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
    dc_cdf[in_pli][id][01] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
        ][id][01][15:1] + 'd02 : dc_cdf[in_pli][id][01]) + ((
        xenc >= 01) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
    dc_cdf[in_pli][id][00] <= #1 (dc_cdf_adapt ? dc_cdf[in_pli
        ][id][00][15:1] + 'd01 : dc_cdf[in_pli][id][00]) + ((
        xenc >= 00) ? OD_ADAPT_HAAR_INCREMENT : 'd0);
  end
end

assign abs_x = in_x[OD_HAAR_BITS-1] ? -in_x : in_x;

assign dc_cdf_curr = dc_cdf[in_pli][id];

assign x_new = daala_min_16(abs_x, 32767);

always_ff @(posedge clk) begin
  if (sof) begin
    ex_q16[0] <= #1 'd32768;
    ex_q16[1] <= #1 'd8;
    ex_q16[2] <= #1 'd8;
  end else if (in_vld && in_rdy) begin
    ex_q16[in_pli] <= #1 ex_q16[in_pli] + ((x_new << 16) -
        ex_q16[in_pli]) >> 2;
  end
end
assign ex_q16_curr = ex_q16[in_pli];

// cdf encode
always_comb begin
  case (1'b1)
    ex_q16_curr[31]: lg = 'd32;
    ex_q16_curr[30]: lg = 'd31;
    ex_q16_curr[29]: lg = 'd30;
    ex_q16_curr[28]: lg = 'd29;
    ex_q16_curr[27]: lg = 'd28;
    ex_q16_curr[26]: lg = 'd27;
    ex_q16_curr[25]: lg = 'd26;
    ex_q16_curr[24]: lg = 'd25;
    ex_q16_curr[23]: lg = 'd24;
    ex_q16_curr[22]: lg = 'd23;
    ex_q16_curr[21]: lg = 'd22;
    ex_q16_curr[20]: lg = 'd21;
    ex_q16_curr[19]: lg = 'd20;
    ex_q16_curr[18]: lg = 'd19;
    ex_q16_curr[17]: lg = 'd18;
```

```
    ex_q16_curr[16]: lg = 'd17;
    ex_q16_curr[15]: lg = 'd16;
    ex_q16_curr[14]: lg = 'd15;
    ex_q16_curr[13]: lg = 'd14;
    ex_q16_curr[12]: lg = 'd13;
    ex_q16_curr[11]: lg = 'd12;
    ex_q16_curr[10]: lg = 'd11;
    ex_q16_curr[09]: lg = 'd10;
    ex_q16_curr[08]: lg = 'd09;
    ex_q16_curr[07]: lg = 'd08;
    ex_q16_curr[06]: lg = 'd07;
    ex_q16_curr[05]: lg = 'd06;
    ex_q16_curr[04]: lg = 'd05;
    ex_q16_curr[03]: lg = 'd04;
    ex_q16_curr[02]: lg = 'd03;
    ex_q16_curr[01]: lg = 'd02;
    ex_q16_curr[00]: lg = 'd01;
    default        : lg = 'd00;
  endcase
end

assign tmp = ex_q16_curr >> (lg - 8);
assign odd = (lg < 15) ? ex_q16_curr*ex_q16_curr > 2 << 2*lg :
   tmp*tmp > (1 << 15);

assign lg_q1 = daala_max_16s(0, 2*lg - 33 + odd);

assign shift = daala_max_16s('d0, (lg_q1 - 5) >> 1);
assign shift_gt_spec = shift > special;

assign id = daala_min_5(11, lg_q1);

assign xs = (abs_x + (1 << shift >> 1)) >> shift;
assign xs_gte_15 = xs >= 15;

assign s = daala_min_5(15, xs);
assign s_gt_0 = s > 0;

assign ft_m1 = dc_cdf_curr[15] - 'd1;

always_comb begin
  case (1'b1)
    ft_m1[13]: s2 = 'd01;
    ft_m1[12]: s2 = 'd02;
    ft_m1[11]: s2 = 'd03;
    ft_m1[10]: s2 = 'd04;
    ft_m1[09]: s2 = 'd05;
```

```
    ft_m1[08]: s2 = 'd06;
    ft_m1[07]: s2 = 'd07;
    ft_m1[06]: s2 = 'd08;
    ft_m1[05]: s2 = 'd09;
    ft_m1[04]: s2 = 'd10;
    ft_m1[03]: s2 = 'd11;
    ft_m1[02]: s2 = 'd12;
    ft_m1[01]: s2 = 'd13;
    ft_m1[00]: s2 = 'd14;
    default  : s2 = 'd15;
  endcase
end


always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    out_vld <= #1 'd0;
  end else if (in_vld && !in_vld_reg) begin
    out_vld <= #1 'd1;
  end else if (out_rdy) begin
    out_vld <= #1 'd0;
  end
end

always_ff @(posedge clk) begin
  if (in_vld && !in_vld_reg) begin
    out_fl <= #1 (s_gt_0 ? dc_cdf_curr[s - 1] : 'd0) << s2;
    out_fh <= #1 dc_cdf_curr[s] << s2;
    out_ft <= #1 dc_cdf_curr[15] << s2;
  end
end

// laplace

always_ff @(posedge clk) begin
  if (in_vld && !in_vld_reg)
    e <= #1 ((ex_q16_curr >> 7) + (1 << shift >> 1)) >> shift;
end

div_e_1 div_e_i (
  .aclk(clk),

  .s_axis_divisor_tvalid  (in_vld_d1),
  .s_axis_divisor_tdata   (e + 'd256),
  .s_axis_dividend_tvalid (in_vld_d1),
  .s_axis_dividend_tdata  ({e, 8'd0}),
```

```
  .m_axis_dout_tvalid     (),
  .m_axis_dout_tdata      (e_calc)
);

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    laplace_vld <= #1 'd0;
  else if (in_vld_d2 && xs_gte_15)
    laplace_vld <= #1 'd1;
  else if (laplace_rdy)
    laplace_vld <= #1 'd0;
end

always_ff @(posedge clk) begin
  if (in_vld_d2) begin
    e_calc_d1 <= #1 e_calc;
  end
end

always_ff @(posedge clk) begin
  if (in_vld_d2) begin
    laplace_x <= #1 xs - 'd15;
    laplace_decay <= #1 daala_max_16('d2, daala_min_16('d254,
       e_calc_d1));
  end
end

// bits
assign special = xs == 0;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    bits_vld_x <= #1 'd0;
  else if (shift_gt_spec && ((in_vld_d2 && !xs_gte_15) || (
     laplace_vld && laplace_rdy)))
    bits_vld_x <= #1 'd1;
  else if (bits_rdy)
    bits_vld_x <= #1 'd0;
end

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    bits_vld_s <= #1 'd0;
  else if ((abs_x != 'd0) && ((in_vld_d2 && !xs_gte_15 && !
     shift_gt_spec) || (laplace_vld && laplace_rdy && !
     shift_gt_spec) || (bits_vld_x && bits_rdy)))
    bits_vld_s <= #1 'd1;
```

```
  else if (bits_rdy)
    bits_vld_s <= #1 'd0;
end
assign bits_vld = bits_vld_x || bits_vld_s;

always_ff @(posedge clk) begin
  bits_ft  <= #1 bits_vld_x ? abs_x - (xs << shift) + (!special
      << (shift - 1)) : in_x[OD_HAAR_BITS-1];
  bits_ftb <= #1 bits_vld_x ? shift - special : 'd1;
end

assign in_rdy = !in_vld || ((abs_x == 'd0) && !shift_gt_spec &&
    !xs_gte_15 && in_vld_d2) || ((abs_x == 'd0) && !
   shift_gt_spec && (laplace_vld && laplace_rdy)) || ((abs_x ==
    'd0) && bits_vld_x && bits_rdy) || (bits_vld_s && bits_rdy)
   ;

endmodule

import daala_pkg::*;

module laplace_enc (
  input  logic        rst,
  input  logic        clk,

  input  logic        in_vld,
  input  logic [15:0] in_x,
  input  logic [15:0] in_decay,
  output logic        in_rdy,

  output logic        out_cdf_vld,
  output logic [15:0] out_cdf_fl,
  output logic [15:0] out_cdf_fh,
  output logic [15:0] out_cdf_ft,
  input  logic        out_cdf_rdy,

  output logic        out_q15_vld,
  output logic [15:0] out_q15_fl,
  output logic [15:0] out_q15_fh,
  input  logic        out_q15_rdy,

  output logic        bits_vld,
  output logic [31:0] bits_ft,
  output logic [ 4:0] bits_ftb,
  input  logic        bits_rdy
);
```

```
logic [8:0] decay;
logic       decay_done;
logic       decay_done_reg;
logic       decay_vld;
logic       decay_stb;
logic [3:0] shift;
logic       shift_eq_0;
logic [7:0] decay_calc;

logic       [ 6:0] cdf_idx;
logic [15:0][15:0] cdf;

logic       in_vld_reg;
logic       in_vld_stb;

logic [3:0] sym;
logic       sym_gt_0;
logic [15:0] xs;
logic [15:0] ms;
logic [3:0]  s;
logic       end_cond;
logic       send_cond;

assign in_vld_stb = in_vld && !in_vld_reg;
always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    in_vld_reg <= #1 'd0;
  end else if (in_rdy) begin
    in_vld_reg <= #1 'd0;
  end else if (in_vld_stb) begin
    in_vld_reg <= #1 'd1;
  end
end

always_ff @(posedge clk) begin
  if (in_vld_stb) begin
    decay <= #1 in_decay;
    shift <= #1 'd0;
  end else if (!decay_done) begin
    decay <= #1 (decay*decay + 128) >> 8;
    shift <= #1 shift + 'd1;
  end
end

assign shift_eq_0 = shift == 'd0;

assign decay_done = !in_vld_stb && (decay <= 235);
```

```
always_ff @(posedge clk, posedge rst) begin
  if (rst)
    decay_done_reg <= #1 'd0;
  else if (in_rdy)
    decay_done_reg <= #1 'd0;
  else if (decay_done)
    decay_done_reg <= #1 'd1;
end

assign decay_calc = daala_max_16(daala_min_16(decay, 254), 2);

assign cdf_idx = (decay + 1) >> 1;

OD_EXP_CDF_TABLE OD_EXP_CDF_TABLE_i (
  .clka (clk),
  .addra(cdf_idx),
  .douta(cdf)
);

always_ff @(posedge clk) begin
  if (decay_stb && !decay_done_reg) begin
    xs <= #1 in_x >> shift;
    ms <= #1  -1 >> shift;
    sym <= #1 daala_min_16(in_x >> shift, 'd15);
  end else if (decay_stb) begin
    sym <= #1 daala_min_16(xs, 'd15);
    xs <= xs - 'd15;
    ms <= ms - 'd15;
  end
end

assign decay_stb = (in_vld && decay_done && !decay_done_reg) ||
    (decay_done_reg && (!decay_vld || !(out_cdf_vld ||
   out_q15_vld) || (out_cdf_rdy && out_q15_rdy)) && !end_cond);

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    decay_vld <= #1 'd0;
  end else if (decay_stb) begin
    decay_vld <= #1 'd1;
  end else if ((out_cdf_rdy && out_q15_rdy) || !(out_cdf_vld ||
      out_q15_vld)) begin
    decay_vld <= #1 'd0;
  end
end
```

```
assign sym_gt_0 = sym > 'd0;
assign end_cond = (sym < 'd15) || (ms == 'd0);
assign send_cond = !ms[15] && (ms < 'd15);

always_comb begin
  case (1'b1)
    ms[13] : s = 'd01;
    ms[12] : s = 'd02;
    ms[11] : s = 'd03;
    ms[10] : s = 'd04;
    ms[09] : s = 'd05;
    ms[08] : s = 'd06;
    ms[07] : s = 'd07;
    ms[06] : s = 'd08;
    ms[05] : s = 'd09;
    ms[04] : s = 'd10;
    ms[03] : s = 'd11;
    ms[02] : s = 'd12;
    ms[01] : s = 'd13;
    ms[00] : s = 'd14;
    default: s = 'd15;
  endcase
end

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    out_cdf_vld <= #1 'd0;
  end else if (decay_vld && send_cond && (!out_cdf_vld ||
     out_cdf_rdy)) begin
    out_cdf_vld <= #1 'd1;
  end else if (out_cdf_rdy) begin
    out_cdf_vld <= #1 'd0;
  end
end

always_ff @(posedge clk) begin
  if (decay_vld && send_cond && (!out_cdf_vld || out_cdf_rdy))
     begin
    out_cdf_fl <= #1 (sym_gt_0 ? cdf[sym - 1] : 'd0) << s;
    out_cdf_fh <= #1 cdf[sym] << s;
    out_cdf_ft <= #1 cdf[ms] << s;
  end
end

always_ff @(posedge clk, posedge rst) begin
```

```
  if (rst) begin
    out_q15_vld <= #1 'd0;
  end else if (decay_vld && !send_cond && (!out_q15_vld ||
     out_q15_rdy)) begin
    out_q15_vld <= #1 'd1;
  end else if (out_q15_rdy) begin
    out_q15_vld <= #1 'd0;
  end
end


always_ff @(posedge clk) begin
  if (decay_vld && !send_cond && (!out_q15_vld || out_q15_rdy))
      begin
    out_q15_fl <= #1 sym_gt_0 ? cdf[sym - 1] : 'd0;
    out_q15_fh <= #1 cdf[sym];
  end
end


always_ff @(posedge clk, posedge rst) begin
  if (rst)
    bits_vld <= #1 'd0;
  else if (decay_done && end_cond && !shift_eq_0 && !bits_vld)
    bits_vld <= #1 'd1;
  else if (bits_rdy)
    bits_vld <= #1 'd0;
end

always_ff @(posedge clk) begin
  bits_ft <= #1 in_x & ((1 << shift) - 1);
  bits_ftb <= shift;
end

assign in_rdy = !in_vld || (decay_done_reg && end_cond &&
   shift_eq_0) || (bits_vld && bits_rdy);

endmodule


module sb_buff (
  input  logic        clk,
  input  logic        rst,

  input  logic        sof,
  input  logic [ 4:0] nhsb,
  input  logic [ 4:0] nvsb,
```

```
  input  logic        haar_done,
  input  logic        haar_en  ,
  input  logic        haar_wea ,
  input  logic        haar_web ,
  input  logic [11:0] haar_adra,
  input  logic [11:0] haar_adrb,
  input  logic [15:0] haar_dina,
  input  logic [15:0] haar_dinb,

  input  logic        max_tree_done,
  input  logic        max_tree_en  ,
  input  logic [11:0] max_tree_adr ,
  output logic [15:0] max_tree_dout,

  input  logic        max_tree_entenc_done,
  input  logic        max_tree_entenc_en  ,
  input  logic [11:0] max_tree_entenc_adr ,
  output logic [15:0] max_tree_entenc_dout,

  output logic        sb_buff_switch
);

logic [1:0] buff_sel;

logic        sb_buff_0_ena  ;
logic        sb_buff_0_wea  ;
logic [11:0] sb_buff_0_addra;
logic [15:0] sb_buff_0_dina ;
logic [15:0] sb_buff_0_douta;
logic        sb_buff_0_enb  ;
logic        sb_buff_0_web  ;
logic [11:0] sb_buff_0_addrb;
logic [15:0] sb_buff_0_dinb ;
logic [15:0] sb_buff_0_doutb;
logic        sb_buff_1_ena  ;
logic        sb_buff_1_wea  ;
logic [11:0] sb_buff_1_addra;
logic [15:0] sb_buff_1_dina ;
logic [15:0] sb_buff_1_douta;
logic        sb_buff_1_enb  ;
logic        sb_buff_1_web  ;
logic [11:0] sb_buff_1_addrb;
logic [15:0] sb_buff_1_dinb ;
logic [15:0] sb_buff_1_doutb;
logic        sb_buff_2_ena  ;
logic        sb_buff_2_wea  ;
logic [11:0] sb_buff_2_addra;
```

```
logic [15:0] sb_buff_2_dina ;
logic [15:0] sb_buff_2_douta;
logic        sb_buff_2_enb  ;
logic        sb_buff_2_web  ;
logic [11:0] sb_buff_2_addrb;
logic [15:0] sb_buff_2_dinb ;
logic [15:0] sb_buff_2_doutb;

logic [11:0] sb_cnt;
logic [11:0] sb_cnt_end_m1;
logic [11:0] sb_cnt_end_m2;

assign sb_cnt_end_m1 = nhsb * nvsb * 'd3 - 'd2;
assign sb_cnt_end_m2 = nhsb * nvsb * 'd3 - 'd3;

always_ff @(posedge clk) begin
  if(sof)
    sb_cnt <= #1 'd0;
  else if (sb_buff_switch)
    sb_cnt <= #1 sb_cnt + 'd1;
end

assign sb_buff_switch = (haar_done || (sb_cnt > sb_cnt_end_m2))
    && (max_tree_done || (sb_cnt < 'd1) || (sb_cnt >
   sb_cnt_end_m1)) && (max_tree_entenc_done || (sb_cnt < 'd2));

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    buff_sel <= #1 'd0;
  end else if (sb_buff_switch) begin
    unique case (buff_sel)
      'd0: buff_sel <= #1 'd1;
      'd1: buff_sel <= #1 'd2;
      'd2: buff_sel <= #1 'd0;
    endcase
  end
end

always_comb begin
  unique case (buff_sel)
    'd0: max_tree_dout = sb_buff_1_douta;
    'd1: max_tree_dout = sb_buff_2_douta;
    'd2: max_tree_dout = sb_buff_0_douta;
  endcase
end

always_comb begin
```

```
  unique case (buff_sel)
    'd0: max_tree_entenc_dout = sb_buff_2_douta;
    'd1: max_tree_entenc_dout = sb_buff_0_douta;
    'd2: max_tree_entenc_dout = sb_buff_1_douta;
  endcase
end

always_comb begin
  unique case (buff_sel)
    'd0: begin
      sb_buff_0_ena   = haar_en;
      sb_buff_0_wea   = haar_wea;
      sb_buff_0_addra = haar_adra;
      sb_buff_0_dina  = haar_dina;

      sb_buff_0_enb   = haar_en;
      sb_buff_0_web   = haar_web;
      sb_buff_0_addrb = haar_adrb;
      sb_buff_0_dinb  = haar_dinb;
    end
    'd1: begin
      sb_buff_0_ena   = max_tree_en;
      sb_buff_0_wea   = 'b0;
      sb_buff_0_addra = max_tree_adr;
      sb_buff_0_dina  = 'd0;

      sb_buff_0_enb   = 'd0;
      sb_buff_0_web   = 'd0;
      sb_buff_0_addrb = 'd0;
      sb_buff_0_dinb  = 'd0;
    end
    'd2: begin
      sb_buff_0_ena   = 'd0;
      sb_buff_0_wea   = 'b0;
      sb_buff_0_addra = 'd0;
      sb_buff_0_dina  = 'd0;

      sb_buff_0_enb   = max_tree_entenc_en;
      sb_buff_0_web   = 'd0;
      sb_buff_0_addrb = max_tree_entenc_adr;
      sb_buff_0_dinb  = 'd0;
    end
  endcase
end

dpram_sb_4096x16 sb_buff_0 (
  .clka   (clk),
```

```
  .clkb   (clk),

  .ena    (sb_buff_0_ena  ),
  .wea    (sb_buff_0_wea  ),
  .addra  (sb_buff_0_addra),
  .dina   (sb_buff_0_dina ),
  .douta  (sb_buff_0_douta),

  .enb    (sb_buff_0_enb  ),
  .web    (sb_buff_0_web  ),
  .addrb  (sb_buff_0_addrb),
  .dinb   (sb_buff_0_dinb ),
  .doutb  (sb_buff_0_doutb)
);


always_comb begin
  unique case (buff_sel)
    'd1: begin
      sb_buff_1_ena   = haar_en;
      sb_buff_1_wea   = haar_wea;
      sb_buff_1_addra = haar_adra;
      sb_buff_1_dina  = haar_dina;

      sb_buff_1_enb   = haar_en;
      sb_buff_1_web   = haar_web;
      sb_buff_1_addrb = haar_adrb;
      sb_buff_1_dinb  = haar_dinb;
    end
    'd2: begin
      sb_buff_1_ena   = max_tree_en;
      sb_buff_1_wea   = 'b0;
      sb_buff_1_addra = max_tree_adr;
      sb_buff_1_dina  = 'd0;

      sb_buff_1_enb   = 'd0;
      sb_buff_1_web   = 'd0;
      sb_buff_1_addrb = 'd0;
      sb_buff_1_dinb  = 'd0;
    end
    'd0: begin
      sb_buff_1_ena   = 'd0;
      sb_buff_1_wea   = 'b0;
      sb_buff_1_addra = 'd0;
      sb_buff_1_dina  = 'd0;

      sb_buff_1_enb   = max_tree_entenc_en;
```

```
      sb_buff_1_web   = 'd0;
      sb_buff_1_addrb = max_tree_entenc_adr;
      sb_buff_1_dinb  = 'd0;
    end
  endcase
end


dpram_sb_4096x16 sb_buff_1 (
  .clka   (clk),
  .clkb   (clk),

  .ena    (sb_buff_1_ena  ),
  .wea    (sb_buff_1_wea  ),
  .addra  (sb_buff_1_addra),
  .dina   (sb_buff_1_dina ),
  .douta  (sb_buff_1_douta),

  .enb    (sb_buff_1_enb  ),
  .web    (sb_buff_1_web  ),
  .addrb  (sb_buff_1_addrb),
  .dinb   (sb_buff_1_dinb ),
  .doutb  (sb_buff_1_doutb)
);

always_comb begin
  unique case (buff_sel)
    'd2: begin
      sb_buff_2_ena   = haar_en;
      sb_buff_2_wea   = haar_wea;
      sb_buff_2_addra = haar_adra;
      sb_buff_2_dina  = haar_dina;

      sb_buff_2_enb   = haar_en;
      sb_buff_2_web   = haar_web;
      sb_buff_2_addrb = haar_adrb;
      sb_buff_2_dinb  = haar_dinb;
    end
    'd0: begin
      sb_buff_2_ena   = max_tree_en;
      sb_buff_2_wea   = 'b0;
      sb_buff_2_addra = max_tree_adr;
      sb_buff_2_dina  = 'd0;

      sb_buff_2_enb   = 'd0;
      sb_buff_2_web   = 'd0;
      sb_buff_2_addrb = 'd0;
```

```
      sb_buff_2_dinb  = 'd0;
    end
    'd1: begin
      sb_buff_2_ena   = 'd0;
      sb_buff_2_wea   = 'b0;
      sb_buff_2_addra = 'd0;
      sb_buff_2_dina  = 'd0;

      sb_buff_2_enb   = max_tree_entenc_en;
      sb_buff_2_web   = 'd0;
      sb_buff_2_addrb = max_tree_entenc_adr;
      sb_buff_2_dinb  = 'd0;
    end
  endcase
end


dpram_sb_4096x16 sb_buff_2 (
  .clka   (clk),
  .clkb   (clk),

  .ena    (sb_buff_2_ena  ),
  .wea    (sb_buff_2_wea  ),
  .addra  (sb_buff_2_addra),
  .dina   (sb_buff_2_dina ),
  .douta  (sb_buff_2_douta),

  .enb    (sb_buff_2_enb  ),
  .web    (sb_buff_2_web  ),
  .addrb  (sb_buff_2_addrb),
  .dinb   (sb_buff_2_dinb ),
  .doutb  (sb_buff_2_doutb)
);

endmodule


module sb_tree_sum_buff (
  input  logic        clk,
  input  logic        rst,

  input  logic        max_tree_done,
  input  logic        max_tree_en  ,
  input  logic [ 1:0] max_tree_wea ,
  input  logic [10:0] max_tree_adra,
  input  logic [10:0] max_tree_adrb,
  input  logic [31:0] max_tree_dina,
  output logic [31:0] max_tree_douta,
```

```
  output logic [31:0] max_tree_doutb,

  input  logic        max_tree_entenc_done,
  input  logic        max_tree_entenc_en  ,
  input  logic [10:0] max_tree_entenc_adra ,
  input  logic [10:0] max_tree_entenc_adrb ,
  output logic [31:0] max_tree_entenc_douta,
  output logic [31:0] max_tree_entenc_doutb
);

logic       buff_sel;

logic        sb_buff_0_ena  ;
logic [ 3:0] sb_buff_0_wea  ;
logic [10:0] sb_buff_0_addra;
logic [31:0] sb_buff_0_dina ;
logic [31:0] sb_buff_0_douta;
logic        sb_buff_0_enb  ;
logic [ 3:0] sb_buff_0_web  ;
logic [10:0] sb_buff_0_addrb;
logic [31:0] sb_buff_0_dinb ;
logic [31:0] sb_buff_0_doutb;

logic        sb_buff_1_ena  ;
logic [ 3:0] sb_buff_1_wea  ;
logic [10:0] sb_buff_1_addra;
logic [31:0] sb_buff_1_dina ;
logic [31:0] sb_buff_1_douta;
logic        sb_buff_1_enb  ;
logic [ 3:0] sb_buff_1_web  ;
logic [10:0] sb_buff_1_addrb;
logic [31:0] sb_buff_1_dinb ;
logic [31:0] sb_buff_1_doutb;

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    buff_sel <= #1 'd0;
  end else if (max_tree_done && max_tree_entenc_done) begin
    buff_sel <= #1 ~buff_sel;
  end
end

always_comb begin
  unique case (buff_sel)
    'd0: max_tree_douta = sb_buff_0_douta;
    'd1: max_tree_douta = sb_buff_1_douta;
  endcase
```

```
  unique case (buff_sel)
    'd0: max_tree_doutb = sb_buff_0_doutb;
    'd1: max_tree_doutb = sb_buff_1_doutb;
  endcase
end

always_comb begin
  unique case (buff_sel)
    'd0: max_tree_entenc_douta = sb_buff_1_douta;
    'd1: max_tree_entenc_douta = sb_buff_0_douta;
  endcase
  unique case (buff_sel)
    'd0: max_tree_entenc_doutb = sb_buff_1_doutb;
    'd1: max_tree_entenc_doutb = sb_buff_0_doutb;
  endcase
end

always_comb begin
  unique case (buff_sel)
    'd0: begin
      sb_buff_0_ena   = max_tree_en;
      sb_buff_0_wea   = {{2{max_tree_wea[1]}}, {2{max_tree_wea
          [0]}}};
      sb_buff_0_addra = max_tree_adra;
      sb_buff_0_dina  = max_tree_dina;

      sb_buff_0_enb   = max_tree_en;
      sb_buff_0_web   = 4'b0000;
      sb_buff_0_addrb = max_tree_adrb;
      sb_buff_0_dinb  = 'd0;
    end
    'd1: begin
      sb_buff_0_ena   = max_tree_entenc_en;
      sb_buff_0_wea   = 'b0;
      sb_buff_0_addra = max_tree_entenc_adra;
      sb_buff_0_dina  = 'd0;

      sb_buff_0_enb   = max_tree_entenc_en;
      sb_buff_0_web   = 'd0;
      sb_buff_0_addrb = max_tree_entenc_adrb;
      sb_buff_0_dinb  = 'd0;
    end
  endcase
end

dpram_sb_2048x32 sb_buff_0 (
  .clka   (clk),
```

```
  .clkb   (clk),

  .ena    (sb_buff_0_ena  ),
  .wea    (sb_buff_0_wea  ),
  .addra  (sb_buff_0_addra),
  .dina   (sb_buff_0_dina ),
  .douta  (sb_buff_0_douta),

  .enb    (sb_buff_0_enb  ),
  .web    (sb_buff_0_web  ),
  .addrb  (sb_buff_0_addrb),
  .dinb   (sb_buff_0_dinb ),
  .doutb  (sb_buff_0_doutb)
);

always_comb begin
  unique case (buff_sel)
    'd1: begin
      sb_buff_1_ena   = max_tree_en;
      sb_buff_1_wea   = {{2{max_tree_wea[1]}}, {2{max_tree_wea
         [0]}}};
      sb_buff_1_addra = max_tree_adra;
      sb_buff_1_dina  = max_tree_dina;

      sb_buff_1_enb   = max_tree_en;
      sb_buff_1_web   = 4'b0000;
      sb_buff_1_addrb = max_tree_adrb;
      sb_buff_1_dinb  = 'd0;
    end
    'd0: begin
      sb_buff_1_ena   = max_tree_entenc_en;
      sb_buff_1_wea   = 'b0;
      sb_buff_1_addra = max_tree_entenc_adra;
      sb_buff_1_dina  = 'd0;

      sb_buff_1_enb   = max_tree_entenc_en;
      sb_buff_1_web   = 'd0;
      sb_buff_1_addrb = max_tree_entenc_adrb;
      sb_buff_1_dinb  = 'd0;
    end
  endcase
end



dpram_sb_2048x32 sb_buff_1 (
  .clka   (clk),
```

```
  .clkb   (clk),

  .ena    (sb_buff_1_ena  ),
  .wea    (sb_buff_1_wea  ),
  .addra  (sb_buff_1_addra),
  .dina   (sb_buff_1_dina ),
  .douta  (sb_buff_1_douta),

  .enb    (sb_buff_1_enb  ),
  .web    (sb_buff_1_web  ),
  .addrb  (sb_buff_1_addrb),
  .dinb   (sb_buff_1_dinb ),
  .doutb  (sb_buff_1_doutb)
);

endmodule


module entenc_buff (
  input  logic        clk,

  input  logic        norm_en,
  input  logic        norm_bs,
  input  logic [15:0] norm_adr,
  input  logic [31:0] norm_dat_w,

  input  logic        done_en,
  input  logic        done_we,
  input  logic [15:0] done_adr,
  input  logic [15:0] done_dat_w,
  output logic [15:0] done_dat_r
);

entenc_pc_buff_1 entenc_pc_buff_i (
 .clka  (clk),
 .ena   (norm_en || done_en),
 .wea   (norm_en ? 1'b1             : done_we),
 .addra (norm_en ? norm_adr         : done_adr),
 .dina  (norm_en ? norm_dat_w[15:0] : done_dat_w),
 .douta (done_dat_r),

 .clkb  (clk),
 .enb   (norm_en && norm_bs),
 .web   (1'b1),
 .addrb (norm_adr + 'd1),
 .dinb  (norm_dat_w[31:16]),
 .doutb ()
);
```

```
endmodule

import daala_pkg::*;

module entenc_calc (
  input  logic rst,
  input  logic clk,

  input  logic [31:0] entenc_low,
  input  logic [15:0] entenc_rng,

  input  logic        in_bool_q15_vld,
  input  logic        in_bool_q15_val,
  input  logic [15:0] in_bool_q15_fz,
  output logic        in_bool_q15_rdy,

  input  logic        in_q15_vld,
  input  logic [15:0] in_q15_fl,
  input  logic [15:0] in_q15_fh,
  output logic        in_q15_rdy,

  input  logic        in_cdf_adapt_vld,
  input  logic [15:0] in_cdf_adapt_fl,
  input  logic [15:0] in_cdf_adapt_fh,
  input  logic [15:0] in_cdf_adapt_ft,
  output logic        in_cdf_adapt_rdy,

  output logic        out_vld,
  output logic [31:0] out_low,
  output logic [15:0] out_rng,
  input  logic        out_rdy
);


// bool_q_15
logic [30:0] bool_q15_v_mul;
logic [15:0] bool_q15_v;

assign bool_q15_v_mul = in_bool_q15_fz * entenc_rng;
assign bool_q15_v = bool_q15_v_mul[30:15];

assign in_bool_q15_rdy = out_rdy;

//q15
logic [30:0] q15_v_mul;
```

```
logic [15:0] q15_v;

logic [30:0] q15_u_mul;
logic [15:0] q15_u;

assign q15_u_mul = in_q15_fl * entenc_rng;
assign q15_v_mul = in_q15_fh * entenc_rng;

assign q15_u = q15_u_mul[30:15];
assign q15_v = q15_v_mul[30:15];

assign in_q15_rdy = out_rdy && !in_bool_q15_vld;

//cdf adapt
logic        cdf_adapt_shift;

logic [15:0] cdf_adapt_fl;
logic [15:0] cdf_adapt_fh;
logic [15:0] cdf_adapt_ft;

logic [15:0] cdf_adapt_d;
logic [15:0] cdf_adapt_d_m2;
logic [15:0] cdf_adapt_e;

logic [15:0] cdf_adapt_fl_e;
logic [15:0] cdf_adapt_fh_e;

logic [15:0] cdf_adapt_u;
logic [15:0] cdf_adapt_v;

assign cdf_adapt_shift = (entenc_rng - in_cdf_adapt_ft) >=
   in_cdf_adapt_ft;

assign cdf_adapt_ft = in_cdf_adapt_ft << cdf_adapt_shift;
assign cdf_adapt_fl = in_cdf_adapt_fl << cdf_adapt_shift;
assign cdf_adapt_fh = in_cdf_adapt_fh << cdf_adapt_shift;

assign cdf_adapt_d = entenc_rng - cdf_adapt_ft;

assign cdf_adapt_d_m2 = cdf_adapt_d*2;

assign cdf_adapt_e = cdf_adapt_d_m2 - daala_min_16(
   cdf_adapt_d_m2, cdf_adapt_ft);

assign cdf_adapt_fl_e = cdf_adapt_fl - daala_min_16(
   cdf_adapt_fl, cdf_adapt_e);
assign cdf_adapt_fh_e = cdf_adapt_fl - daala_min_16(
```

```
    cdf_adapt_fh, cdf_adapt_e);

assign cdf_adapt_u = cdf_adapt_fl + daala_min_16(cdf_adapt_fl,
    cdf_adapt_e) + daala_min_16(cdf_adapt_fl_e >> 1, cdf_adapt_d
    );
assign cdf_adapt_v = cdf_adapt_fh + daala_min_16(cdf_adapt_fh,
    cdf_adapt_e) + daala_min_16(cdf_adapt_fh_e >> 1, cdf_adapt_d
    );


assign in_cdf_adapt_rdy = out_rdy && !in_bool_q15_vld && !
    in_q15_vld;

//out
always_ff @(posedge clk) begin
  if (out_rdy) begin
    case (1'b1)
      in_bool_q15_vld: begin
        out_low <= #1 in_bool_q15_val ? entenc_low + bool_q15_v
            : entenc_low;
        out_rng <= #1 in_bool_q15_val ? entenc_rng - bool_q15_v
            : bool_q15_v;
      end
      in_q15_vld: begin
        out_low <= #1 entenc_low + q15_u;
        out_rng <= #1 q15_v - q15_u;
      end
      in_cdf_adapt_vld: begin
        out_low <= #1 entenc_low + cdf_adapt_u;
        out_rng <= #1 cdf_adapt_v - cdf_adapt_u;
      end
    endcase
  end
end

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else if (out_rdy)
    out_vld <= #1 in_bool_q15_vld || in_q15_vld ||
       in_cdf_adapt_vld;
end

endmodule


import daala_pkg::*;
```

```
module entenc_cdf_adapt (
  input  logic        rst,
  input  logic        clk,

  input  logic        sof,

  input  logic        in_vld,
  input  logic [ 3:0] in_val,
  input  logic [ 1:0] in_cdf_sel,
  input  logic [ 8:0] in_cdf_num,
  input  logic [ 4:0] in_nsyms,
  output logic        in_rdy,

  output logic        out_vld,
  output logic [15:0] out_fl,
  output logic [15:0] out_fh,
  output logic [15:0] out_ft,
  input  logic        out_rdy
);


logic [270+150+3-1:0][15:0][15:0] haar_cdf;
logic [15:0][15:0] haar_cdf_curr;

logic [2:00][8:0] haar_cdf_off;
logic       [8:0] haar_cdf_idx;
logic             haar_cdf_adapt;

assign haar_cdf_off[0] = 'd0;
assign haar_cdf_off[1] = 'd270;
assign haar_cdf_off[2] = 'd270 + 'd150;

assign in_rdy = out_rdy || !out_vld;

logic [ 3:0] in_val_d1;
logic [ 4:0] in_nsyms_d1;

logic [15:0] ft_m1;
logic [ 3:0] s;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else if (out_rdy || !out_vld)
    out_vld <= #1 in_vld;
end
```

```
assign haar_cdf_idx = haar_cdf_off[in_cdf_sel]+in_cdf_num;
assign haar_cdf_adapt = haar_cdf_curr[in_nsyms-1] > 'd32767 -
   OD_ADAPT_HAAR_INCREMENT;

always_ff @(posedge clk) begin
  if (sof)
    haar_cdf  <= #1 {270+150+3{16'd00512, 16'd00480, 16'd00448,
        16'd00416, 16'd00384, 16'd00352, 16'd00320, 16'd00288,
       16'd00256, 16'd00224, 16'd00192, 16'd00160, 16'd00128,
       16'd00096, 16'd00064, 16'd00032}};
  else if (in_vld && in_rdy) begin
    haar_cdf[haar_cdf_idx][15] <= #1 (in_nsyms < 15) ? (
       haar_cdf_adapt ? haar_cdf_curr[15][15:1] + 'd16 :
       haar_cdf_curr[15]) + ((in_val >= 15) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[15];
    haar_cdf[haar_cdf_idx][14] <= #1 (in_nsyms < 14) ? (
       haar_cdf_adapt ? haar_cdf_curr[14][15:1] + 'd15 :
       haar_cdf_curr[14]) + ((in_val >= 14) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[14];
    haar_cdf[haar_cdf_idx][13] <= #1 (in_nsyms < 13) ? (
       haar_cdf_adapt ? haar_cdf_curr[13][15:1] + 'd14 :
       haar_cdf_curr[13]) + ((in_val >= 13) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[13];
    haar_cdf[haar_cdf_idx][12] <= #1 (in_nsyms < 12) ? (
       haar_cdf_adapt ? haar_cdf_curr[12][15:1] + 'd13 :
       haar_cdf_curr[12]) + ((in_val >= 12) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[12];
    haar_cdf[haar_cdf_idx][11] <= #1 (in_nsyms < 11) ? (
       haar_cdf_adapt ? haar_cdf_curr[11][15:1] + 'd12 :
       haar_cdf_curr[11]) + ((in_val >= 11) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[11];
    haar_cdf[haar_cdf_idx][10] <= #1 (in_nsyms < 10) ? (
       haar_cdf_adapt ? haar_cdf_curr[10][15:1] + 'd11 :
       haar_cdf_curr[10]) + ((in_val >= 10) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[10];
    haar_cdf[haar_cdf_idx][09] <= #1 (in_nsyms < 09) ? (
       haar_cdf_adapt ? haar_cdf_curr[09][15:1] + 'd10 :
       haar_cdf_curr[09]) + ((in_val >= 09) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[09];
    haar_cdf[haar_cdf_idx][08] <= #1 (in_nsyms < 08) ? (
       haar_cdf_adapt ? haar_cdf_curr[08][15:1] + 'd09 :
       haar_cdf_curr[08]) + ((in_val >= 08) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[08];
    haar_cdf[haar_cdf_idx][07] <= #1 (in_nsyms < 07) ? (
       haar_cdf_adapt ? haar_cdf_curr[07][15:1] + 'd08 :
       haar_cdf_curr[07]) + ((in_val >= 07) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[07];
```

```
    haar_cdf[haar_cdf_idx][06] <= #1 (in_nsyms < 06) ? (
       haar_cdf_adapt ? haar_cdf_curr[06][15:1] + 'd07 :
       haar_cdf_curr[06]) + ((in_val >= 06) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[06];
    haar_cdf[haar_cdf_idx][05] <= #1 (in_nsyms < 05) ? (
       haar_cdf_adapt ? haar_cdf_curr[05][15:1] + 'd06 :
       haar_cdf_curr[05]) + ((in_val >= 05) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[05];
    haar_cdf[haar_cdf_idx][04] <= #1 (in_nsyms < 04) ? (
       haar_cdf_adapt ? haar_cdf_curr[04][15:1] + 'd05 :
       haar_cdf_curr[04]) + ((in_val >= 04) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[04];
    haar_cdf[haar_cdf_idx][03] <= #1 (in_nsyms < 03) ? (
       haar_cdf_adapt ? haar_cdf_curr[03][15:1] + 'd04 :
       haar_cdf_curr[03]) + ((in_val >= 03) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[03];
    haar_cdf[haar_cdf_idx][02] <= #1 (in_nsyms < 02) ? (
       haar_cdf_adapt ? haar_cdf_curr[02][15:1] + 'd03 :
       haar_cdf_curr[02]) + ((in_val >= 02) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[02];
    haar_cdf[haar_cdf_idx][01] <= #1 (in_nsyms < 01) ? (
       haar_cdf_adapt ? haar_cdf_curr[01][15:1] + 'd02 :
       haar_cdf_curr[01]) + ((in_val >= 01) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[01];
    haar_cdf[haar_cdf_idx][00] <= #1 (in_nsyms < 00) ? (
       haar_cdf_adapt ? haar_cdf_curr[00][15:1] + 'd01 :
       haar_cdf_curr[00]) + ((in_val >= 00) ?
       OD_ADAPT_HAAR_INCREMENT : 'd0) : haar_cdf_curr[00];
  end
end

assign haar_cdf_curr = haar_cdf[haar_cdf_idx];

assign ft_m1 = haar_cdf_curr[in_nsyms - 1] - 'd1;

always_comb begin
  case (1'b1)
    ft_m1[13]: s = 'd01;
    ft_m1[12]: s = 'd02;
    ft_m1[11]: s = 'd03;
    ft_m1[10]: s = 'd04;
    ft_m1[09]: s = 'd05;
    ft_m1[08]: s = 'd06;
    ft_m1[07]: s = 'd07;
    ft_m1[06]: s = 'd08;
    ft_m1[05]: s = 'd09;
    ft_m1[04]: s = 'd10;
```

```
    ft_m1[03]: s = 'd11;
    ft_m1[02]: s = 'd12;
    ft_m1[01]: s = 'd13;
    ft_m1[00]: s = 'd14;
    default  : s = 'd15;
  endcase
end



always_ff @(posedge clk) begin
  if (in_vld && in_rdy) begin
    out_fl <= #1 ((in_val > 0) ? haar_cdf_curr[in_val - 1] : '
       d0) << s;
    out_fh <= #1 haar_cdf_curr[in_val] << s;
    out_ft <= #1 haar_cdf_curr[in_nsyms - 1] << s;
  end
end

endmodule


module entenc_cdf_adapt_switch (
  input  logic        rst,
  input  logic        clk,

  input  logic        in_gen_enc_vld,
  input  logic [15:0] in_gen_enc_fl ,
  input  logic [15:0] in_gen_enc_fh ,
  input  logic [15:0] in_gen_enc_ft ,
  output logic        in_gen_enc_rdy,

  input  logic        in_laplace_enc_vld,
  input  logic [15:0] in_laplace_enc_fl ,
  input  logic [15:0] in_laplace_enc_fh ,
  input  logic [15:0] in_laplace_enc_ft ,
  output logic        in_laplace_enc_rdy,

  input  logic        in_max_tree_vld,
  input  logic [15:0] in_max_tree_fl ,
  input  logic [15:0] in_max_tree_fh ,
  input  logic [15:0] in_max_tree_ft ,
  output logic        in_max_tree_rdy,

  output logic        out_vld,
  output logic [15:0] out_fl ,
  output logic [15:0] out_fh ,
  output logic [15:0] out_ft ,
```

```
  input  logic        out_rdy

);


always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else
    out_vld <= #1 in_gen_enc_vld || in_laplace_enc_vld ||
       in_max_tree_vld;
end

always_ff @(posedge clk) begin
  case (1'b1)
    in_max_tree_vld: begin
      out_fl <= #1 in_max_tree_fl;
      out_fh <= #1 in_max_tree_fl;
      out_ft <= #1 in_max_tree_ft;
    end
    in_gen_enc_vld: begin
      out_fl <= #1 in_gen_enc_fl;
      out_fh <= #1 in_gen_enc_fh;
      out_ft <= #1 in_gen_enc_ft;
    end
    in_laplace_enc_vld: begin
      out_fl <= #1 in_laplace_enc_fl;
      out_fh <= #1 in_laplace_enc_fh;
      out_ft <= #1 in_laplace_enc_ft;
    end
  endcase
end

always_comb begin
  in_gen_enc_rdy = 1'b0;
  in_laplace_enc_rdy = 1'b0;
  in_max_tree_rdy = 1'b0;
  case (1'b1)
    in_max_tree_vld: begin
      in_max_tree_rdy = 1'b1;
    end
    in_gen_enc_vld: begin
      in_gen_enc_rdy = 1'b1;
    end
    in_laplace_enc_vld: begin
      in_laplace_enc_rdy = 1'b1;
    end
```

```
    default: begin
      in_gen_enc_rdy = 1'b1;
      in_laplace_enc_rdy = 1'b1;
      in_max_tree_rdy = 1'b1;
    end
  endcase
end


endmodule


import daala_pkg::*;

module entenc_ctrl (
  input logic         clk,
  input logic         rst,

  input logic         sof,

  output logic        bool_q15_vld,
  output logic        bool_q15_val,
  output logic [15:0] bool_q15_fz,
  input  logic        bool_q15_rdy,

  output logic        uint_vld,
  output logic [31:0] uint_fl,
  output logic [31:0] uint_ft,
  input  logic        uint_rdy,

  output logic        bits_vld,
  output logic [31:0] bits_fl,
  output logic [ 4:0] bits_ftb,
  input  logic        bits_rdy
);

localparam [3:0] STATE_INIT       = 0;
localparam [3:0] STATE_DATA_PACK  = 1;
localparam [3:0] STATE_KEYFRAME   = 2;
localparam [3:0] STATE_FRAME_NUM  = 3;
localparam [3:0] STATE_AM         = 4;
localparam [3:0] STATE_QM         = 5;
localparam [3:0] STATE_HAAR       = 6;
localparam [3:0] STATE_GOLDEN     = 7;
localparam [3:0] STATE_PVQ        = 8;
localparam [3:0] STATE_CODED_QT   = 9;
localparam [3:0] STATE_WAIT       = 10;
```

```
logic       out_trn;

logic [3:0] state;
logic [7:0] state_cnt;

logic [ 3:0] frame_cnt;

assign out_trn = (bool_q15_vld && bool_q15_rdy) || (uint_vld &&
    uint_rdy) || (bits_vld && bits_rdy) || (!bool_q15_vld && !
   uint_vld && !bits_vld);

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    frame_cnt <= #1 'hf;
  else if (sof)
    frame_cnt <= #1 frame_cnt + 'd1;
end

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    state <= #1 STATE_INIT;
  else if (out_trn) begin
    unique case (state)
      STATE_INIT      : state <= #1 sof ? STATE_DATA_PACK :
         STATE_INIT;
      STATE_DATA_PACK : state <= #1 STATE_KEYFRAME;
      STATE_KEYFRAME  : state <= #1 STATE_FRAME_NUM;
      STATE_FRAME_NUM : state <= #1 STATE_WAIT;
      STATE_WAIT      : state <= #1 (state_cnt == 'd1) ?
         STATE_AM : STATE_WAIT;
      STATE_AM        : state <= #1 STATE_QM;
      STATE_QM        : state <= #1 STATE_HAAR;
      STATE_HAAR      : state <= #1 STATE_GOLDEN;
      STATE_GOLDEN    : state <= #1 STATE_PVQ;
      STATE_PVQ       : state <= #1 (state_cnt == 'd89) ?
         STATE_CODED_QT : STATE_PVQ;
      STATE_CODED_QT  : state <= #1 STATE_INIT;
    endcase
  end
end

always_ff @(posedge clk) begin
  if (out_trn) begin
    case (state)
      STATE_FRAME_NUM ,
      STATE_PVQ       : state_cnt <= #1 state_cnt + 'd1;
      default         : state_cnt <= #1 'd0;
```

```
    endcase
  end
end



// bool q15
always_comb begin
  case (state)
    STATE_DATA_PACK ,
    STATE_KEYFRAME  ,
    STATE_AM        ,
    STATE_QM        ,
    STATE_HAAR      ,
    STATE_GOLDEN    : bool_q15_vld = 1'd1;
    default         : bool_q15_vld = 1'd0;
  endcase
end

always_comb begin
  case (state)
    STATE_DATA_PACK ,
    STATE_AM        : bool_q15_val = 1'b0;
    default         : bool_q15_val = 1'd1;
  endcase
end

assign bool_q15_fz = 16'd16384;

// uint
always_comb begin
  case (state)
    STATE_FRAME_NUM ,
    STATE_CODED_QT  : uint_vld = 1'd1;
    default         : uint_vld = 1'd0;
  endcase
end

always_comb begin
  case (state)
    STATE_FRAME_NUM : uint_fl = frame_cnt;
    default         : uint_fl = 'd0;
  endcase
end

always_comb begin
```

```
  case (state)
    STATE_FRAME_NUM : uint_ft = 'd16;
    default         : uint_ft = 'd64;
  endcase
end

// bits
always_comb begin
  case (state)
    STATE_PVQ : bits_vld = 1'd1;
    default   : bits_vld = 1'd0;
  endcase
end

assign bits_fl = PVQ_INIT[state_cnt];
assign bits_ftb = 8;

endmodule

import daala_pkg::*;

module entenc_norm (
  input  logic        rst,
  input  logic        clk,

  input  logic        in_vld,
  input  logic [31:0] in_low,
  input  logic [15:0] in_rng,
  output logic        in_rdy,

  output logic        pc_buff_en,
  output logic [15:0] pc_buff_adr,
  output logic        pc_buff_bs,
  output logic [31:0] pc_buff_dat_w,

  output logic [31:0] low,
  output logic [15:0] rng,
  output logic [15:0] offs,
  output logic [4:0]  cnt
);



logic        in_vld_reg;

logic [ 4:0] d;
logic [ 4:0] s;
```

```
logic        s_gte_0;
logic        s_gte_8;
logic [ 5:0] c;
logic [31:0] m;
logic [31:0] m8;


assign in_rdy = !in_vld || in_vld_reg;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    in_vld_reg <= #1 'd0;
  else  if (in_rdy)
    in_vld_reg <= #1 'd0;
  else  if (in_vld)
    in_vld_reg <= #1 'd1;
end

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    low        <= 'd0;
    rng        <= 'h8000;
    cnt        <= -'d9;
    offs       <= 'd0;
  end else begin
    if (in_vld && !in_vld_reg) begin
      rng  <= #1 in_rng << d;
      low  <= #1 (in_low & (s_gte_8 ? m8 : (s_gte_0 ? m :
         {32{1'b1}}))) << d;

      cnt  <= #1 s    - (s_gte_8 ? 'd16 : (s_gte_0 ? 'd8 : 'd0)
         );
      offs <= #1 offs + (s_gte_8 ?  'd2 : (s_gte_0 ? 'd1 : 'd0)
         );
    end
  end
end


always_comb begin
  case (1'b1)
    in_rng[15]: d = 'd00;
    in_rng[14]: d = 'd01;
    in_rng[13]: d = 'd02;
    in_rng[12]: d = 'd03;
    in_rng[11]: d = 'd04;
    in_rng[10]: d = 'd05;
```

```
    in_rng[09]: d = 'd06;
    in_rng[08]: d = 'd07;
    in_rng[07]: d = 'd08;
    in_rng[06]: d = 'd09;
    in_rng[05]: d = 'd10;
    in_rng[04]: d = 'd11;
    in_rng[03]: d = 'd12;
    in_rng[02]: d = 'd13;
    in_rng[01]: d = 'd14;
    in_rng[00]: d = 'd15;
    default   : d = 'd16;
  endcase
end

assign s = cnt + d;
assign s_gte_0 = !s[4];
assign s_gte_8 = s_gte_0 && (s >= 'd8);

assign c = {cnt[4], cnt} + 6'd16;

assign m  = (1 <<       c) - 1;
assign m8 = (1 << (c-'d8)) - 1;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    pc_buff_en <= #1 'd0;
  else if (in_vld && !in_vld_reg && s_gte_0)
    pc_buff_en <= #1 'd1;
  else
    pc_buff_en <= #1 'd0;
end

always_ff @(posedge clk) begin
  pc_buff_adr <= #1 offs;
  pc_buff_bs <= #1 s_gte_8;
  pc_buff_dat_w[15: 0] <= #1 (in_low >> c);
  pc_buff_dat_w[31:16] <= #1 (in_low&m >> c-8);
end

endmodule


import daala_pkg::*;

module entenc_uint (
  input  logic rst,
  input  logic clk,
```

```
  input  logic        in_vld,
  input  logic [31:0] in_fl,
  input  logic [31:0] in_ft,
  output logic        in_rdy,

  output logic        out_vld,
  output logic [15:0] out_fl,
  output logic [15:0] out_fh,
  input  logic        out_rdy,

  output logic        bits_vld,
  output logic [31:0] bits_fl,
  output logic [ 4:0] bits_ftb,
  input  logic        bits_rdy
);

logic        ft_gt_16;
logic [31:0] ft_m1;
logic [ 4:0] ftb;
logic [ 4:0] ft1;
logic [ 7:0] cdf_off;
logic [ 4:0] s;

assign in_rdy = (out_rdy || !out_vld) && (bits_rdy || !bits_vld
   );

assign ft_gt_16 = in_ft > 16;
assign ft_m1 = in_ft - 'd1;

always_comb begin
  case (1'b1)
    ft_m1[31]: ftb = 'd28;
    ft_m1[30]: ftb = 'd27;
    ft_m1[29]: ftb = 'd26;
    ft_m1[28]: ftb = 'd25;
    ft_m1[27]: ftb = 'd24;
    ft_m1[26]: ftb = 'd23;
    ft_m1[25]: ftb = 'd22;
    ft_m1[24]: ftb = 'd21;
    ft_m1[23]: ftb = 'd20;
    ft_m1[22]: ftb = 'd19;
    ft_m1[21]: ftb = 'd18;
    ft_m1[20]: ftb = 'd17;
    ft_m1[19]: ftb = 'd16;
    ft_m1[18]: ftb = 'd15;
    ft_m1[17]: ftb = 'd14;
    ft_m1[16]: ftb = 'd13;
```

```
    ft_m1[15]: ftb = 'd12;
    ft_m1[14]: ftb = 'd11;
    ft_m1[13]: ftb = 'd10;
    ft_m1[12]: ftb = 'd09;
    ft_m1[11]: ftb = 'd08;
    ft_m1[10]: ftb = 'd07;
    ft_m1[09]: ftb = 'd16;
    ft_m1[08]: ftb = 'd05;
    ft_m1[07]: ftb = 'd04;
    ft_m1[06]: ftb = 'd03;
    ft_m1[05]: ftb = 'd02;
    ft_m1[04]: ftb = 'd01;
    default  : ftb = 'd00;
  endcase
end


assign ft1 = (ft_m1 >> ftb) + 'd1;

assign s = ft_gt_16 ? in_fl >> ftb : in_fl;

always_comb begin
  unique case (ft_gt_16 ? ft1 : in_ft)
    5'd02: cdf_off = 'd000;
    5'd03: cdf_off = 'd002;
    5'd04: cdf_off = 'd005;
    5'd05: cdf_off = 'd009;
    5'd06: cdf_off = 'd014;
    5'd07: cdf_off = 'd020;
    5'd08: cdf_off = 'd027;
    5'd09: cdf_off = 'd035;
    5'd10: cdf_off = 'd044;
    5'd11: cdf_off = 'd054;
    5'd12: cdf_off = 'd065;
    5'd13: cdf_off = 'd077;
    5'd14: cdf_off = 'd090;
    5'd15: cdf_off = 'd104;
    5'd16: cdf_off = 'd119;
  endcase
end

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else if (out_rdy || !out_vld)
    out_vld <= #1 in_vld;
end
```

```
always_ff @(posedge clk) begin
  out_fl <= #1 s > 0 ? OD_UNIFORM_CDFS_Q15[cdf_off + s - 1] : '
     d0;
  out_fh <= #1 OD_UNIFORM_CDFS_Q15[cdf_off + s];
end

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    bits_vld <= #1 'd0;
  else if (bits_rdy)
    bits_vld <= #1 in_vld && ft_gt_16;
end

always_ff @(posedge clk) begin
  bits_fl  <= #1 in_fl & ((1 << ftb) - 'd1);
  bits_ftb <= #1 ftb;
end

endmodule


module entenc_bits_switch (
  input  logic        rst,
  input  logic        clk,

  input  logic        in_ctrl_vld     ,
  input  logic [31:0] in_ctrl_fl      ,
  input  logic [ 4:0] in_ctrl_ftb     ,
  output logic        in_ctrl_rdy     ,

  input  logic        in_max_tree_vld ,
  input  logic [31:0] in_max_tree_fl  ,
  input  logic [ 4:0] in_max_tree_ftb ,
  output logic        in_max_tree_rdy ,

  input  logic        in_uint_vld     ,
  input  logic [31:0] in_uint_fl      ,
  input  logic [ 4:0] in_uint_ftb     ,
  output logic        in_uint_rdy     ,

  input  logic        in_gen_enc_vld     ,
  input  logic [31:0] in_gen_enc_fl      ,
  input  logic [ 4:0] in_gen_enc_ftb     ,
  output logic        in_gen_enc_rdy     ,

  input  logic        in_laplace_enc_vld     ,
  input  logic [31:0] in_laplace_enc_fl      ,
```

```
  input  logic [ 4:0] in_laplace_enc_ftb     ,
  output logic        in_laplace_enc_rdy     ,

  output logic        out_vld         ,
  output logic [31:0] out_fl          ,
  output logic [ 4:0] out_ftb         ,
  input  logic        out_rdy
);


always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else if (out_rdy)
    out_vld <= #1 in_ctrl_vld || in_uint_vld || in_max_tree_vld
        || in_gen_enc_vld || in_laplace_enc_vld;
end

always_ff @(posedge clk) begin
  if (out_rdy) begin
    case (1'b1)
      in_ctrl_vld: begin
        out_fl <= #1 in_ctrl_fl;
        out_ftb <= #1 in_ctrl_ftb;
      end
      in_uint_vld   : begin
        out_fl <= #1 in_uint_fl;
        out_ftb <= #1 in_uint_ftb;
      end
      in_max_tree_vld: begin
        out_fl <= #1 in_max_tree_fl;
        out_ftb <= #1 in_max_tree_ftb;
      end
      in_gen_enc_vld: begin
        out_fl <= #1 in_gen_enc_fl;
        out_ftb <= #1 in_gen_enc_ftb;
      end
      in_laplace_enc_vld: begin
        out_fl <= #1 in_laplace_enc_fl;
        out_ftb <= #1 in_laplace_enc_ftb;
      end
    endcase
  end
end

always_comb begin
  in_ctrl_rdy = 1'b0;
```

```
  in_uint_rdy = 1'b0;
  in_gen_enc_rdy = 1'b0;
  in_laplace_enc_rdy = 1'b0;
  in_max_tree_rdy = 1'b0;
  case (1'b1)
    in_ctrl_vld: begin
      in_ctrl_rdy = 1'b1;
    end
    in_uint_vld   : begin
      in_uint_rdy = 1'b1;
    end
    in_max_tree_vld: begin
      in_max_tree_rdy = 1'b1;
    end
    in_gen_enc_vld: begin
      in_gen_enc_rdy = 1'b1;
    end
    in_laplace_enc_vld: begin
      in_laplace_enc_rdy = 1'b1;
    end
    default: begin
      in_ctrl_rdy = 1'b1;
      in_uint_rdy = 1'b1;
      in_gen_enc_rdy = 1'b1;
      in_laplace_enc_rdy = 1'b1;
      in_max_tree_rdy = 1'b1;
    end
  endcase
end

endmodule


import daala_pkg::*;

module entenc_bits (
  input  logic rst,
  input  logic clk,

  output logic [31:0]  end_window,
  output logic [ 5:0]  nend_bits,

  input  logic         in_vld,
  input  logic [31:0] in_fl,
  input  logic [ 4:0] in_ftb,
  output logic         in_rdy,

  output logic         out_vld,
```

```
  output logic [31:0] out_dat,
  output logic [ 3:0] out_keep,
  input  logic        out_rdy
);


logic         nend_bits_ftb_gt;
logic [ 2:0]  nend_bits_div_8;

always_ff @(posedge clk, posedge rst) begin
  if (rst) begin
    end_window <= #1 'd0;
    nend_bits  <= #1 'd0;
  end else begin
    if (in_vld) begin
      nend_bits  <= #1 nend_bits - {nend_bits_div_8, 3'd0} +
         in_ftb;
      end_window <= #1 (end_window >> {nend_bits_div_8, 3'b0})
         | (in_fl << (nend_bits - {nend_bits_div_8, 3'd0}));
    end
  end
end

assign nend_bits_ftb_gt = nend_bits + in_ftb >
   OD_EC_WINDOW_SIZE;
assign nend_bits_div_8 = nend_bits_ftb_gt ? nend_bits[5:3] : '
   d0;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else if (out_rdy)
    out_vld <= #1 in_vld && (nend_bits_div_8 > 'd0);
end

always_ff @(posedge clk) begin
  if (in_vld) begin
    out_dat <= #1 end_window;
    out_keep <= #1 (nend_bits_div_8 == 'd1) ? 4'b0001 :
                  ((nend_bits_div_8 == 'd2) ? 4'b0011 :
                  ((nend_bits_div_8 == 'd3) ? 4'b0111 :
                                              4'b1111));
  end
end

assign in_rdy = out_rdy;
```

```
endmodule

import daala_pkg::*;

module entenc_done (
  input  logic        rst,
  input  logic        clk,

  input  logic        in_vld,
  input  logic [31:0] in_dat,
  input  logic [ 3:0] in_keep,
  output logic        in_rdy,

  input  logic        eof_vld,
  input  logic [31:0] eof_low,
  input  logic [15:0] eof_rng,
  input  logic [ 4:0] eof_cnt,
  input  logic [15:0] eof_offs,
  input  logic [ 5:0] eof_nend_bits,
  input  logic [31:0] eof_end_window,

  output logic        pc_buff_en,
  output logic        pc_buff_we,
  output logic [15:0] pc_buff_adr,
  output logic [15:0] pc_buff_dat_w,
  input  logic [15:0] pc_buff_dat_r,

  output logic        out_vld,
  output logic [31:0] out_dat,
  output logic [ 3:0] out_keep,
  output logic        out_lst,
  input  logic        out_rdy
);

localparam [2:0] STATE_FLUSH_INIT    = 'd0;
localparam [2:0] STATE_FLUSH_PREP_PC = 'd1;
localparam [2:0] STATE_FLUSH_PC      = 'd2;
localparam [2:0] STATE_FLUSH_RAW     = 'd3;
localparam [2:0] STATE_FLUSH_BORDER  = 'd4;
localparam [2:0] STATE_FLUSH_CARRY   = 'd5;

logic [ 6:0] s;
logic [31:0] m;
logic [31:0] end_window;
logic [15:0] offs;
logic [31:0] n;
logic [31:0] cnt;
```

```
logic [31:0] nend_bits;
logic [ 2:0] nend_bits_flush;

logic [2:0] state;

assign in_rdy = out_rdy;

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    state <= #1 STATE_FLUSH_INIT;
  else if (out_rdy) begin
    unique case (state)
      STATE_FLUSH_INIT   : state <= #1 eof_vld ?
         STATE_FLUSH_PREP_PC : STATE_FLUSH_INIT;
      STATE_FLUSH_PREP_PC: state <= #1 ((end_window | m) < (
         eof_low + eof_rng)) ? STATE_FLUSH_PC :
         STATE_FLUSH_PREP_PC ;
      STATE_FLUSH_PC     : state <= #1 (s <= 8) ?
         STATE_FLUSH_RAW : STATE_FLUSH_PC;
      STATE_FLUSH_RAW    : state <= #1 STATE_FLUSH_BORDER;
      STATE_FLUSH_BORDER : state <= #1 STATE_FLUSH_CARRY;
      STATE_FLUSH_CARRY  : state <= #1 (offs == 'd1) ?
         STATE_FLUSH_INIT : STATE_FLUSH_CARRY;
    endcase
  end
end
always_ff @(posedge clk) begin
  if (eof_vld && (state == STATE_FLUSH_INIT)) begin
    s <= #1 'd9 + eof_cnt;
    m <= #1 'h7FFF;
    end_window <= #1 (eof_low + 'h7FFF) & ~'hFFFF8000;
    offs <= #1 eof_offs;
    n <= #1 (1 << (eof_cnt + 'd16)) - 1;
    cnt <= #1 eof_cnt;
    nend_bits <= #1 eof_nend_bits;
  end else if (state == STATE_FLUSH_PREP_PC) begin
    s <= #1 s + 'd1;
    m <= #1 m >> 1;
    end_window <= #1 (eof_low + (m>>1)) & ~(m>>1);
  end else if (state == STATE_FLUSH_PC) begin
    end_window <= #1 end_window & n;
    s <= #1 s - 8;
    cnt <= #1 cnt - 8;
    n <= #1 n >> 8;
    offs <= #1 offs + 'd1;
    nend_bits_flush <= #1 (eof_nend_bits > (s + 'd16)) ? 'd4 :
                          ((eof_nend_bits > (s + 'd08)) ? 'd3 :
```

```
                        ((eof_nend_bits > (s       )) ? 'd2 :
                        ((eof_nend_bits > (s - 'd08)) ? 'd1 :
                                                        'd0)));
  end else if (state == STATE_FLUSH_RAW) begin
    cnt <= #1 'd0;
    offs <= #1 offs - 'd1;
    nend_bits <= #1 nend_bits - (nend_bits_flush<<3);
  end else if (state == STATE_FLUSH_BORDER) begin
    offs <= #1 offs - 'd1;
    cnt <= #1 (cnt + pc_buff_dat_r) >> 8;
  end else if (state == STATE_FLUSH_CARRY) begin
    offs <= #1 offs - 'd1;
    cnt <= #1 (cnt + pc_buff_dat_r) >> 8;
  end
end

always_ff @(posedge clk, posedge rst) begin
  if (rst)
    pc_buff_en <= #1 'd0;
  else begin
    case (state)
      STATE_FLUSH_PC     ,
      STATE_FLUSH_RAW    ,
      STATE_FLUSH_BORDER ,
      STATE_FLUSH_CARRY  : pc_buff_en <= #1 'd1;
      default            : pc_buff_en <= #1 'd0;
    endcase
  end
end

always_ff @(posedge clk) begin
  if (state == STATE_FLUSH_PC) begin
    pc_buff_adr <= #1 offs;
    pc_buff_dat_w <= #1 (end_window >> (cnt + 'd16));
    pc_buff_we <= #1 'd1;
  end else begin
    pc_buff_adr <= #1 offs;
    pc_buff_we <= #1 'd0;
  end
end


always_ff @(posedge clk, posedge rst) begin
  if (rst)
    out_vld <= #1 'd0;
  else if (out_rdy)
    out_vld <= #1 in_vld || (state == STATE_FLUSH_RAW) || (
```

```
        state == STATE_FLUSH_BORDER) || (state ==
        STATE_FLUSH_CARRY);
end


always_ff @(posedge clk) begin
  if (in_vld) begin
    out_dat <= #1 in_dat;
    out_keep <= #1 in_keep;
    out_lst <= #1 'd0;
  end else if (state == STATE_FLUSH_RAW) begin
    out_dat <= #1 eof_end_window;
    out_keep <= #1 (1 << nend_bits_flush) - 'd1;
    out_lst <= #1 'd0;
  end else if (state == STATE_FLUSH_BORDER) begin
    out_dat <= #1 ((nend_bits > 0) ? (eof_end_window >> (
       nend_bits_flush<<3)) : 'd0) | pc_buff_dat_r[7:0];
    out_keep <= #1 4'b0001;
    out_lst <= #1 'd0;
  end else if (state == STATE_FLUSH_CARRY) begin
    out_dat <= #1 cnt + pc_buff_dat_r;
    out_keep <= #1 4'b0001;
    out_lst <= #1 (offs == 'd1);
  end
end


endmodule
```

Slike

Tabele

Literatura

[1] "AMBA Specifications - ARM", Arm.com, 2016. [Internet]. Dostopno: http://www.arm.com/products/system-ip/amba-specifications.php. [Dostop: 03- Aug- 2016].

[2] S. Aramvith and M. Sun, "MPEG-1 and MPEG-2 video standards", Handbook of Image and Video Processing, pp. 597–610, 1999.

[3] J. Bankoski, P. Wilkins and Y. Xu, "Technical overview of VP8, an open source video codec for the web.", v ICME, 2011, pp. 1–6.

[4] R. Burns, Television : an international history of the formative years. 1998.

[5] "Encoder triggers an assertion when run with -z 1 · Issue #183 · xiph/daala", GitHub - Daala Issue 183, 2016. [Internet]. Dostopno: https://github.com/xiph/daala/issues/183. [Dostop: 03- Aug- 2016].

[6] D. Grois, M. Detlev, A. Mulayoff, B. Itzhaky and O. Hadar, "Performance comparison of h. 265/mpeg-hevc, vp9, and h. 264/mpeg-avc encoders", v Picture Coding Symposium (PCS), 2013, pp. 394–397.

[7] M. Jacobs and J. Probell, "A brief history of video coding", ARC International, 2007.

[8] D. Mukherjee, J. Bankoski, A. Grange, H. Jingning, J. Koleszar, P. Wilkins, Y. Xu and R. Bultje, "The latest open-source video codec VP9-an overview and preliminary results", v Picture Coding Symposium (PCS), 2013, pp. 390–393.

[9] M. Sharabayko, "Next Generation Video Codecs: HEVC, VP9 and DAALA.", Traffic, vol. 2560, no. 1600, p. 30, 2013.

[10] T. Sikora, "MPEG digital video-coding standards", IEEE Signal Process. Mag., vol. 14, no. 5, pp. 82-100, 1997.

[11] Sony Corporation of America et al. v. Universal City Studios, Inc., et al.. 1984.

[12]]R. Stanković and B. Falkowski, "The Haar wavelet transform: its status and achievements", Computers & Electrical Engineering, vol. 29, no. 1, pp. 25-44, 2003.

[13] G. Sullivan, "Overview of international video coding standards (preceding H. 264/AVC)", v ITU-T VICA workshop, Geneva, 2005.

[14] G. Sullivan, J. Ohm, W. Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1649-1668, 2012.

[15] "Theora.org :: main - Theora, video for everyone", Theora.org, 2016. [Internet]. Dostopno: http://theora.org. [Dostop: 01- Aug- 2016].

[16] T. Wiegand, G. Sullivan, G. Bjontegaard and A. Luthra, "Overview of the H.264/AVC video coding standard", IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 560-576, 2003.

[17] Xilinx Vivado. Xilinx, 2016.

[18] "Xiph.org - daala.git repository", Git.xiph.org, 2016. [Internet]. Dostopno: https://git.xiph.org/?p=daala.git. [Dostop: 01- Aug- 2016].

[19] "Xiph.org :: daala video", Xiph.org, 2016. [Internet]. Dostopno: http://xiph.org/daala. [Dostop: 01- Aug- 2016].